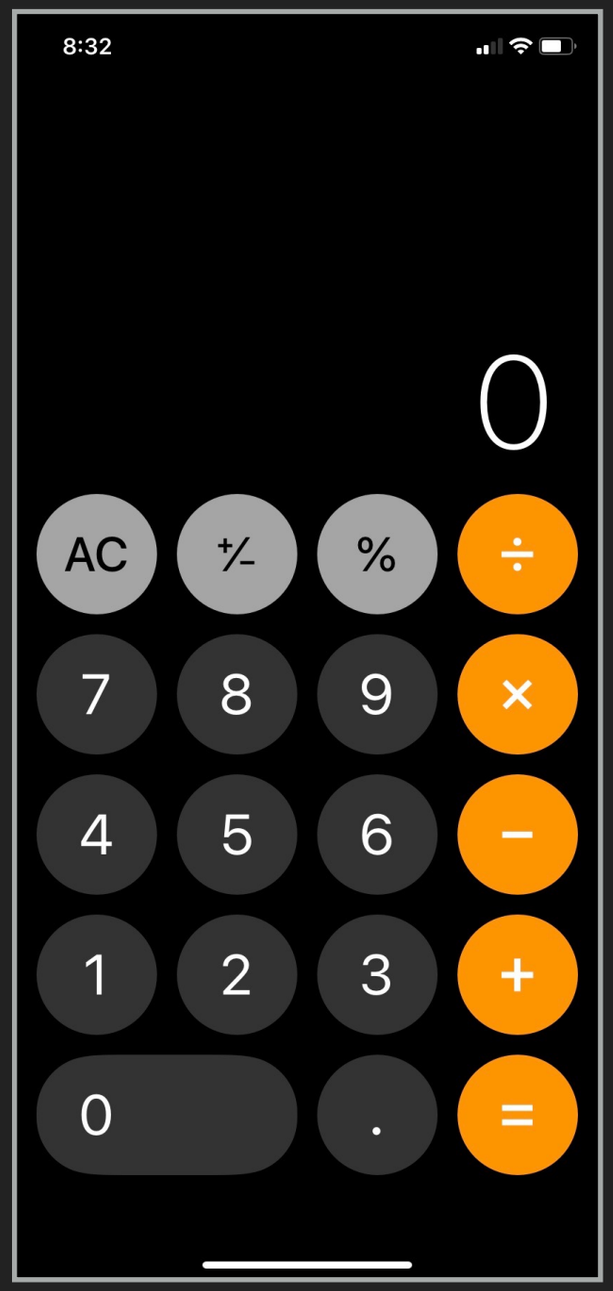
SESSION 3

iOS DEVELOPMENT

MORE TYPES OF

VIEW CONTROLLERS

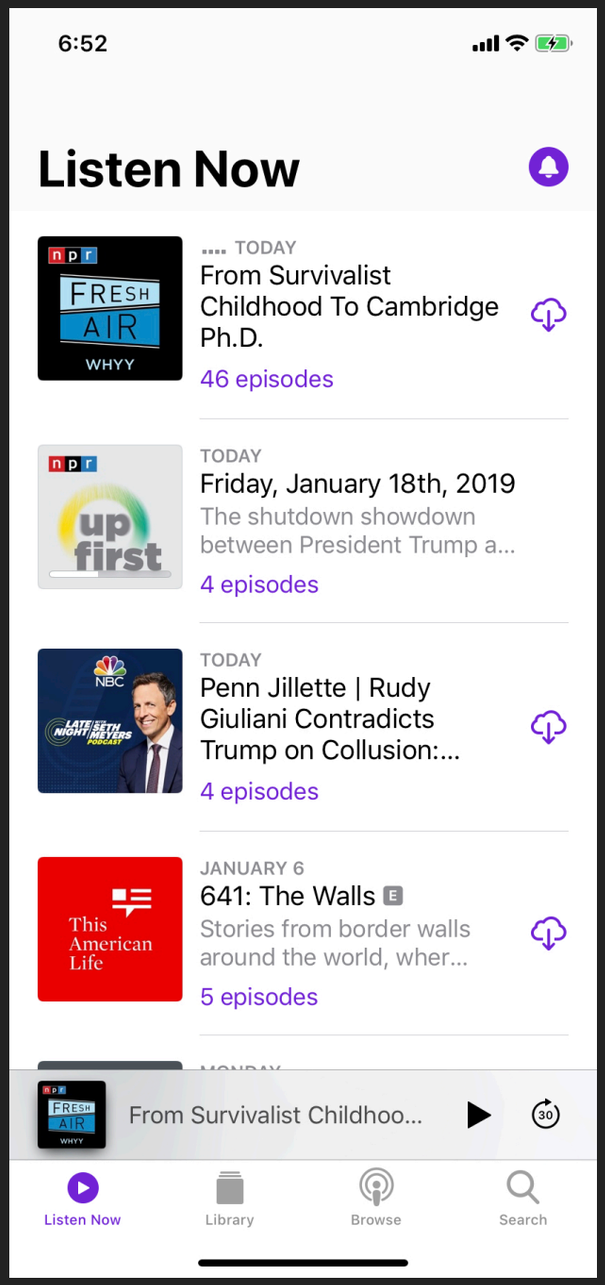
APPS CAN HAVE JUST ONE VIEW CONTROLLER. . .

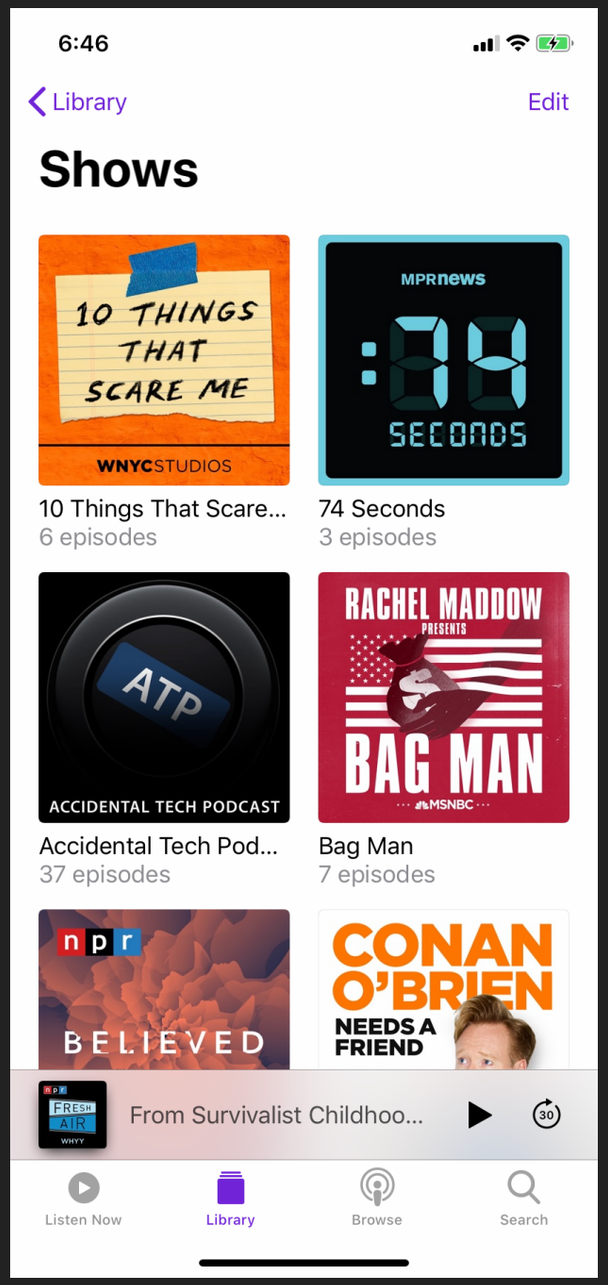


Calculator

Compass

. . . OR MULTIPLE VIEW CONTROLLERS



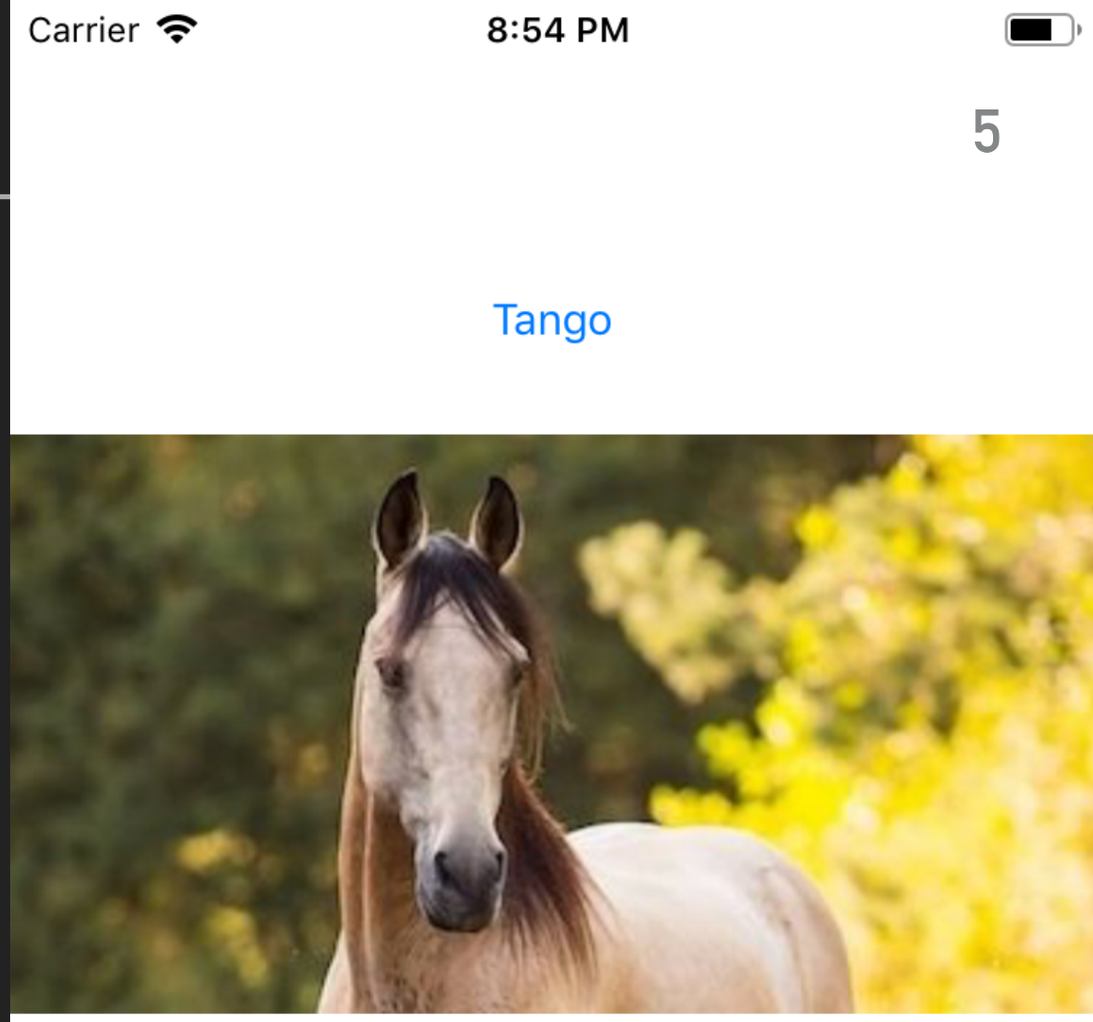


Podcasts

POP QUIZ*

- How many view controllers did your assignment have?

*Worth zero points.



THE VIEW CONTROLLER FAMILY

- UITableViewController
- UICollectionViewController
- UINavigationController
- UITabBarController
- UIPageViewController
- UISplitViewController

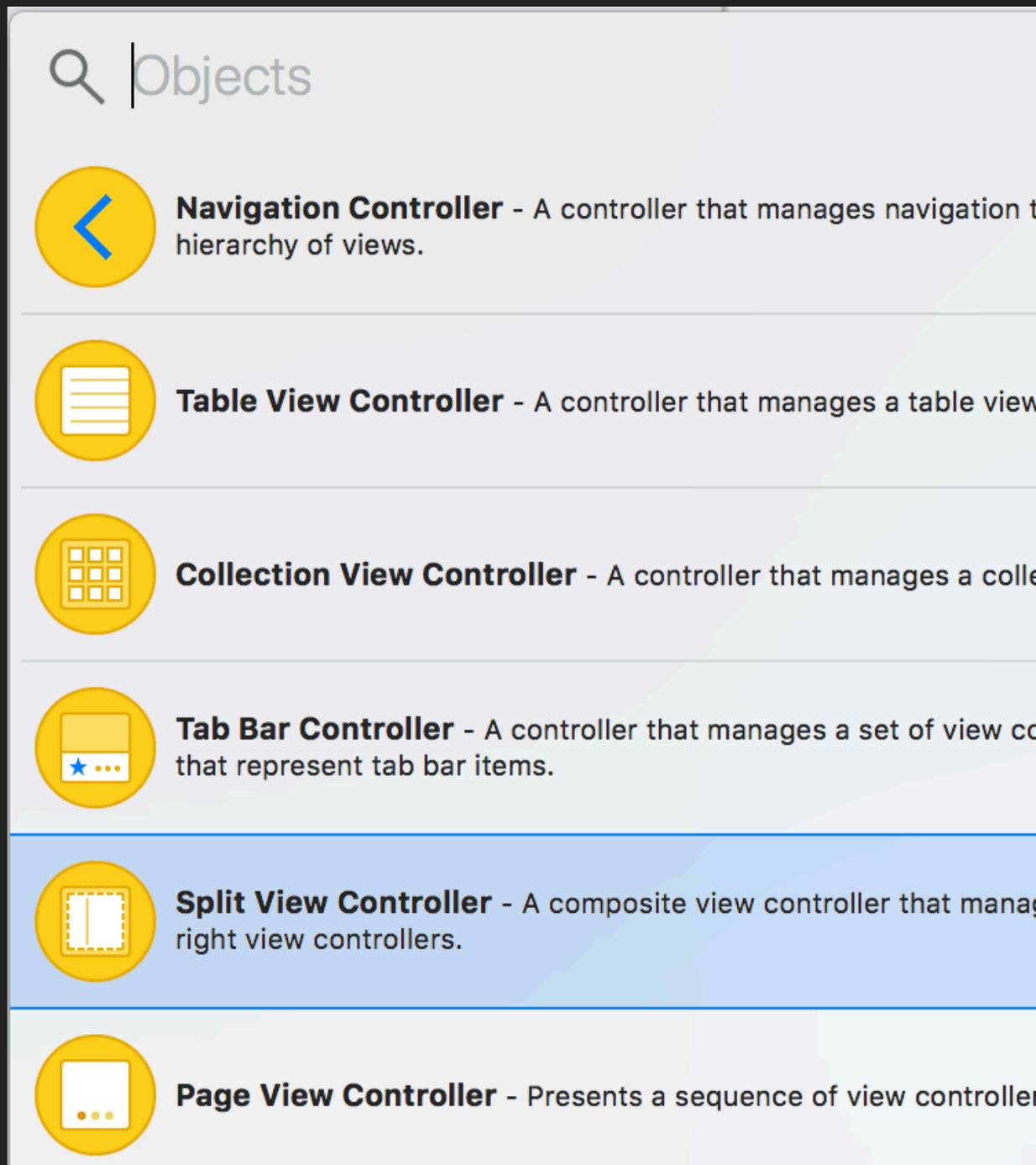
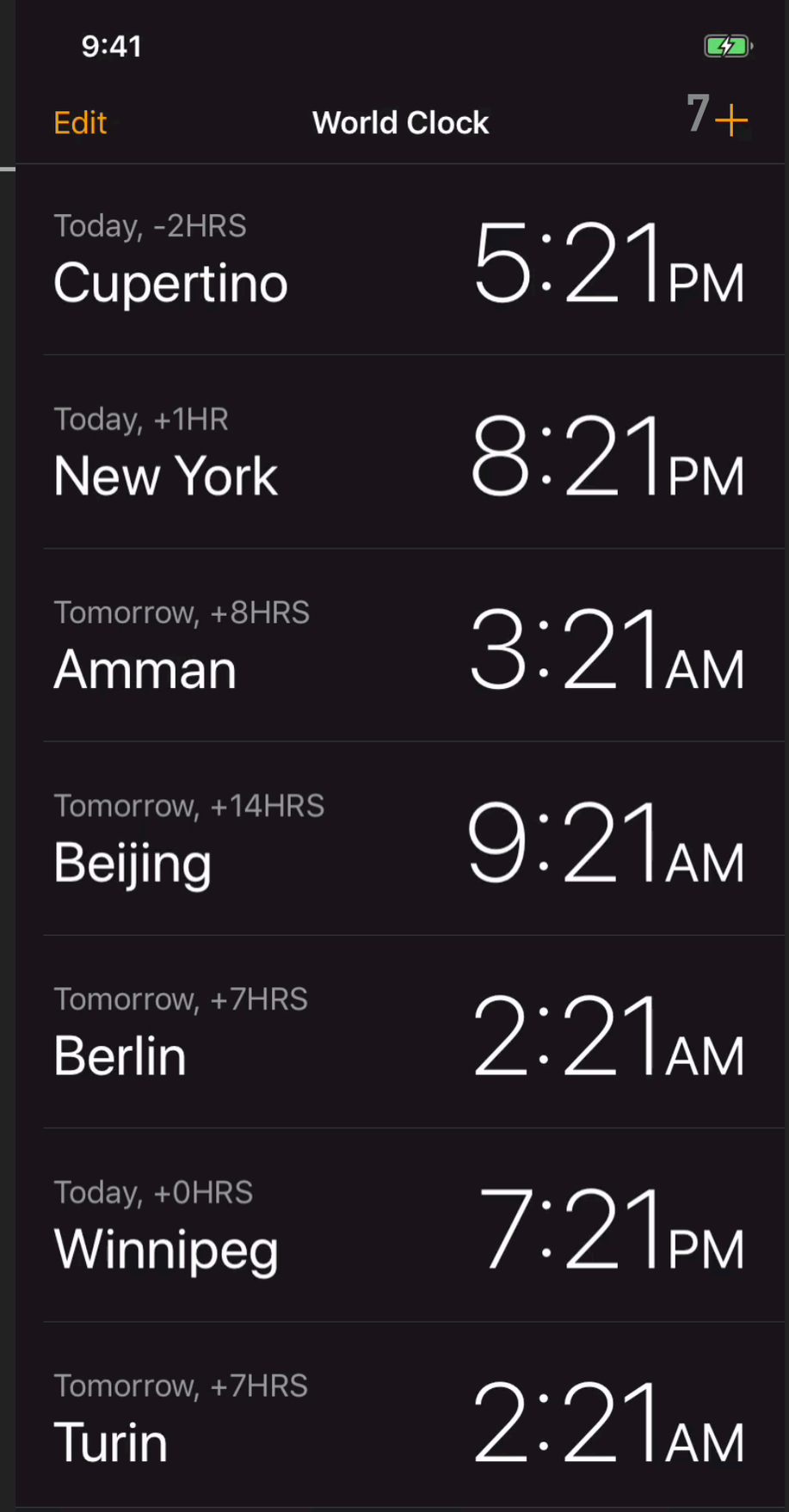


TABLE VIEW CONTROLLER

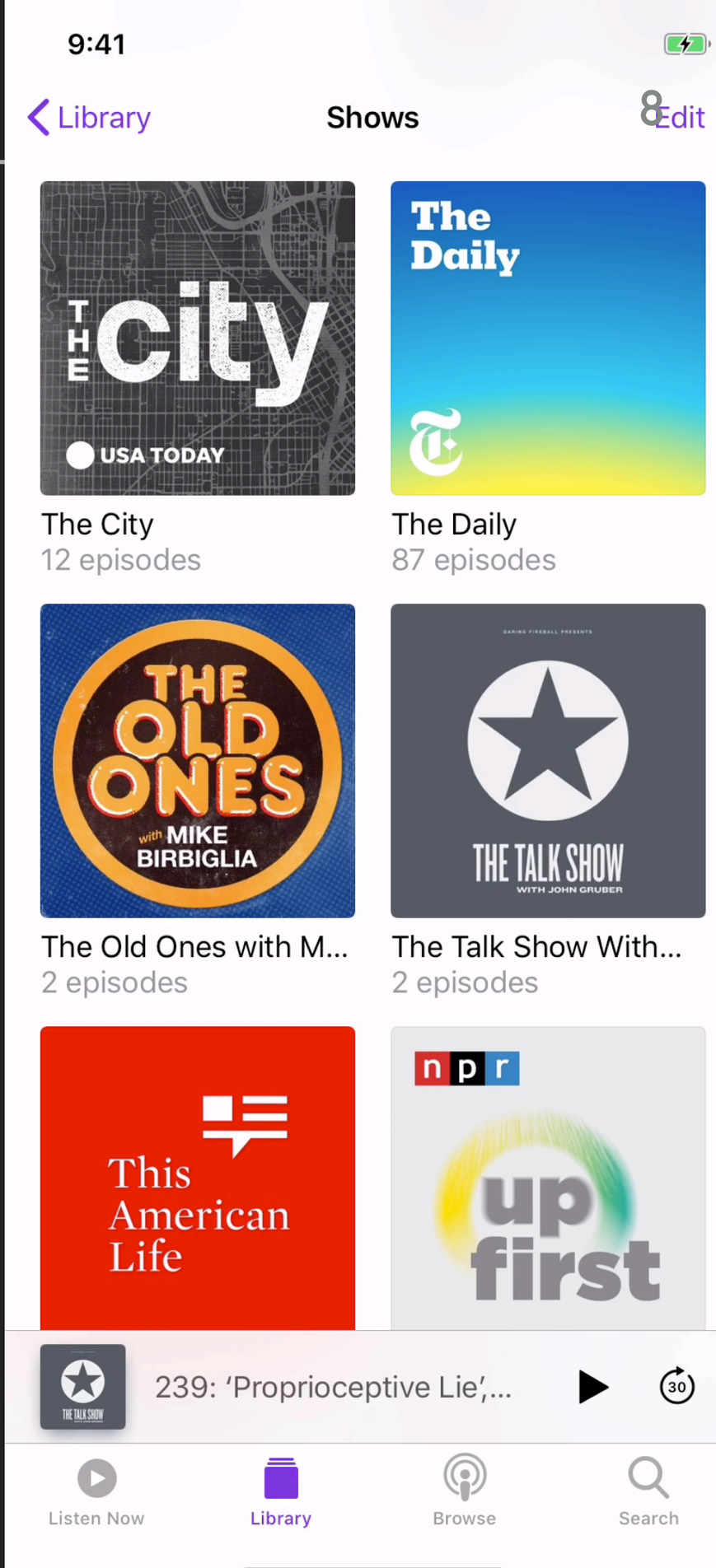
- Manages a table view
- Rows of vertically scrolling content



Clock

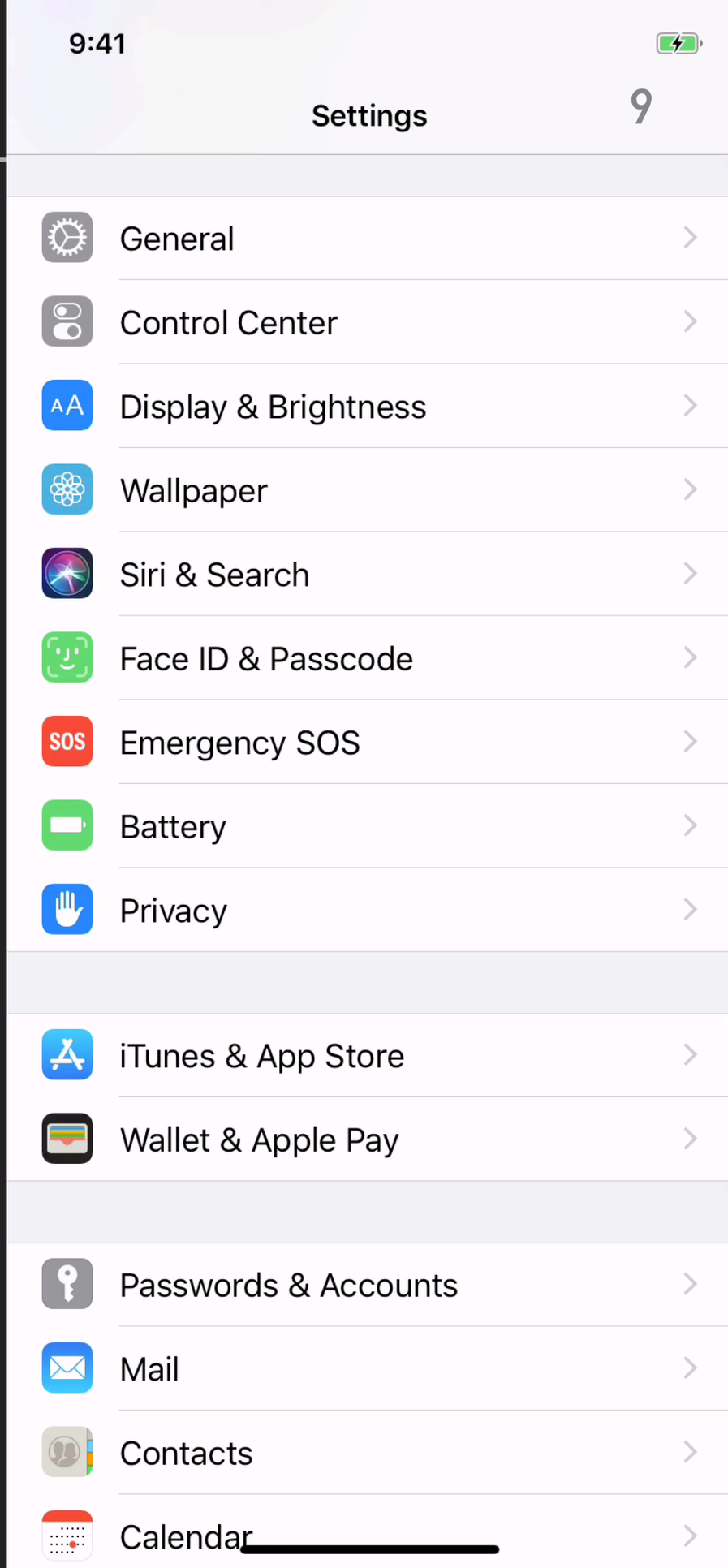
COLLECTION VIEW CONTROLLER

- Manages a collection view
- Rows and columns of vertically or horizontally scrolling content



NAVIGATION CONTROLLER

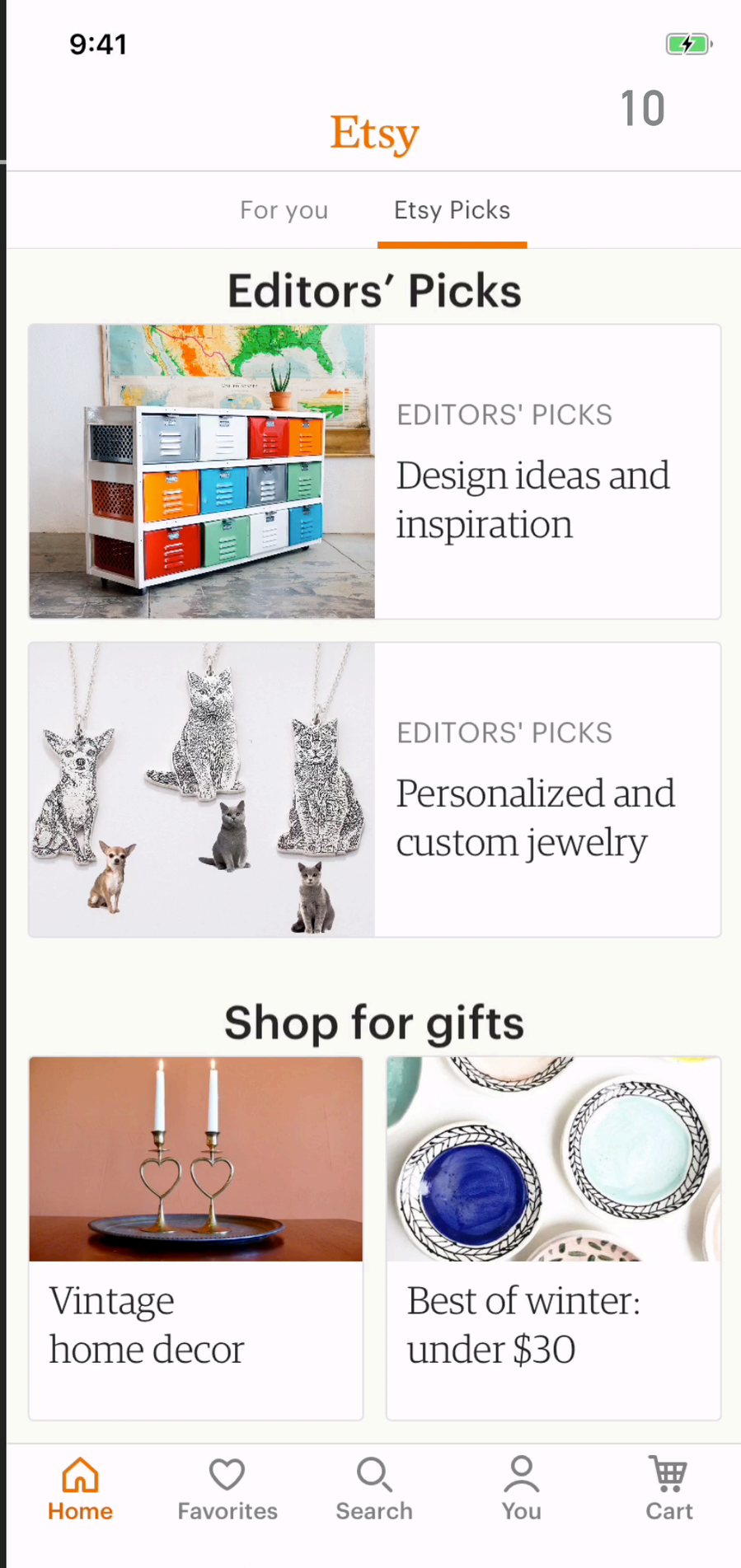
- Navigation bar at the top
- Manages a stack of child view controllers
- Coordinates the transitions between child view controllers



Settings

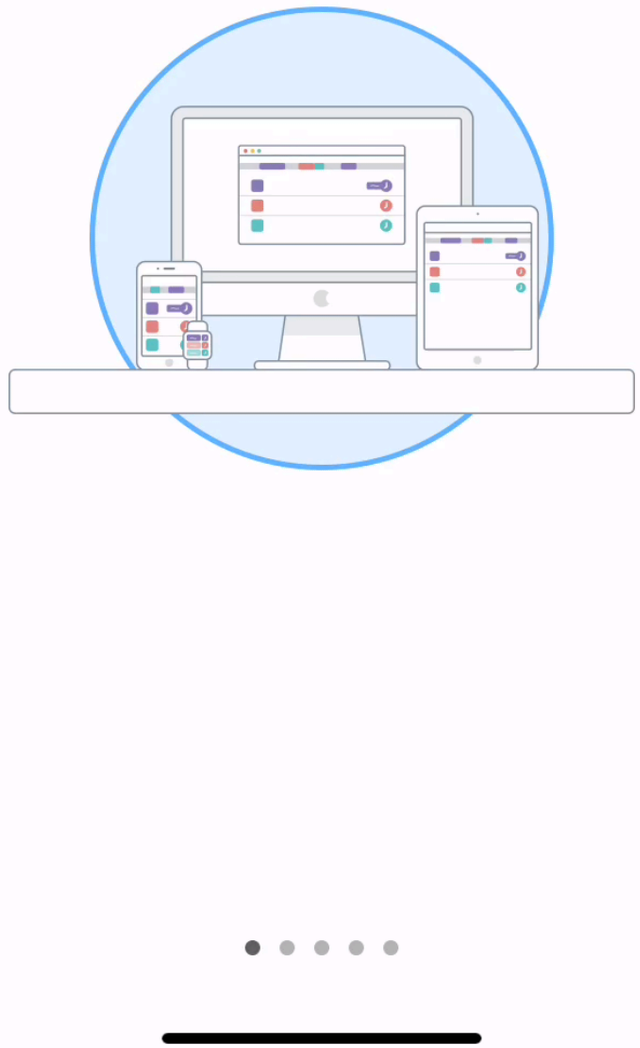
TAB BAR CONTROLLER

- Tab bar at the bottom
- Manages an array of child view controllers, displayed one at a time in any order



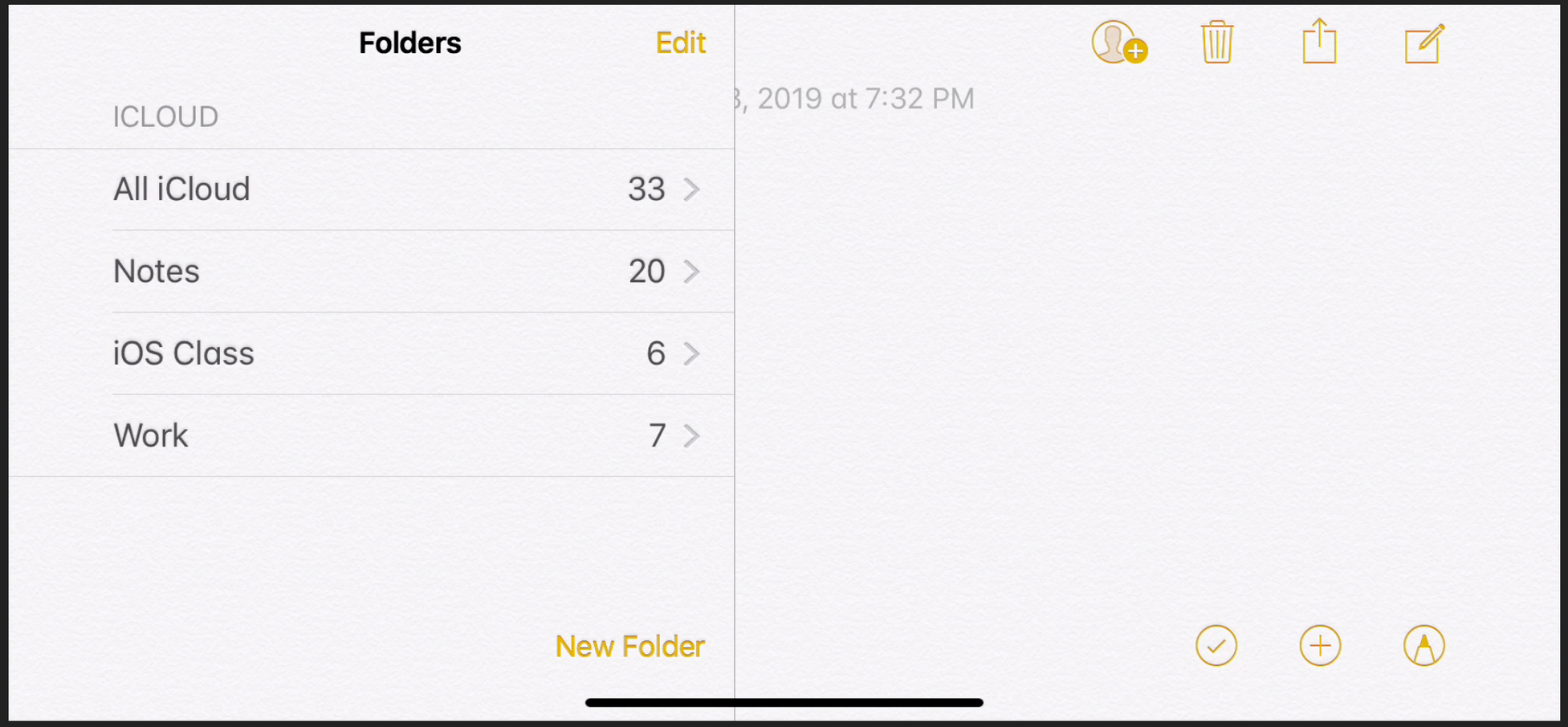
PAGE VIEW CONTROLLER

- Page control at bottom
- Manages an array of child view controllers, displayed one at a time in sequential order



SPLIT VIEW CONTROLLER

- Manages a master view controller (left) and a detail view controller (right)

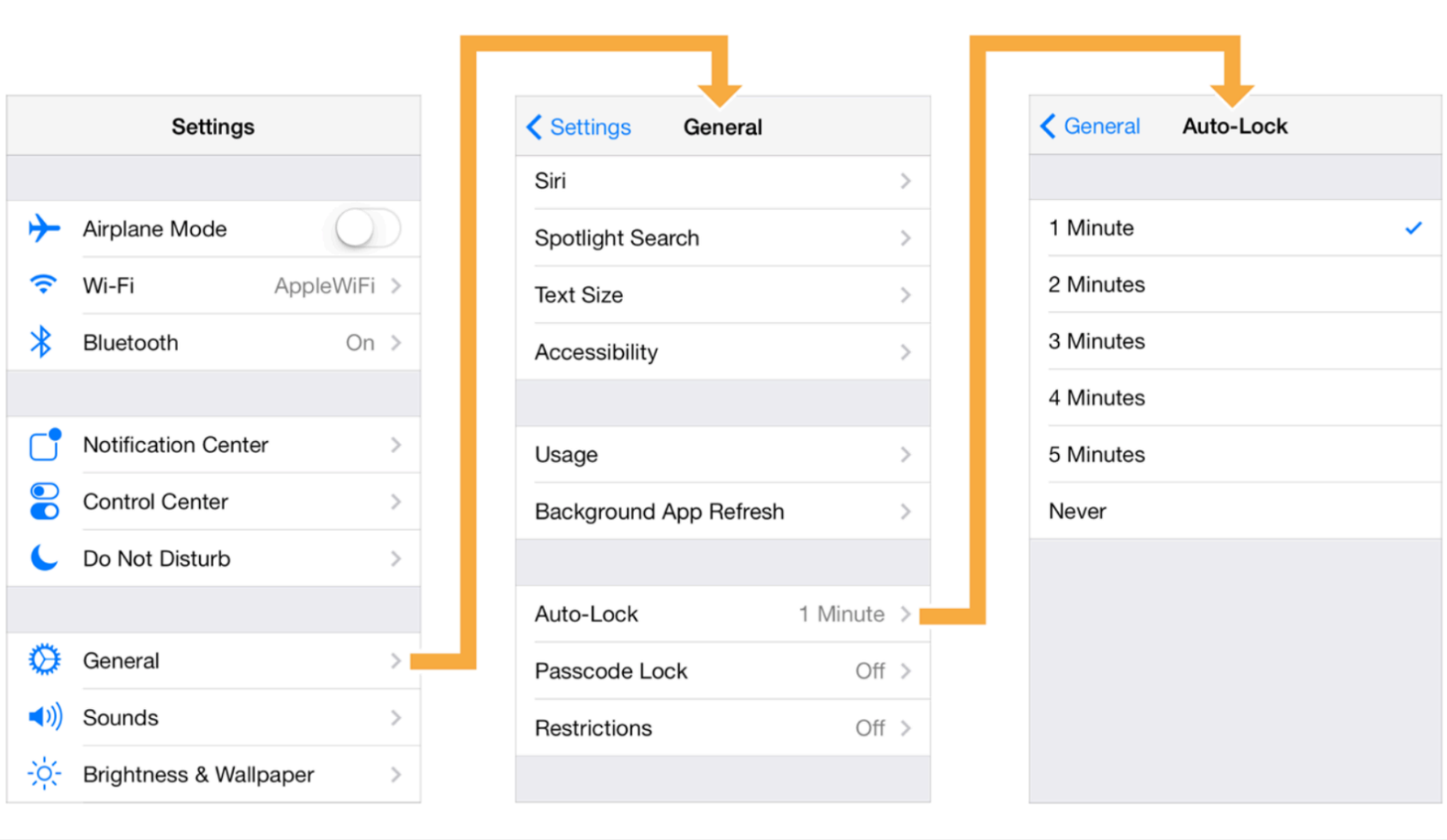


NAVIGATION CONTROLLER

NAVIGATION CONTROLLER

- A container view controller that defines a stack-based scheme for navigating hierarchical content.

Figure 1 A sample navigation interface

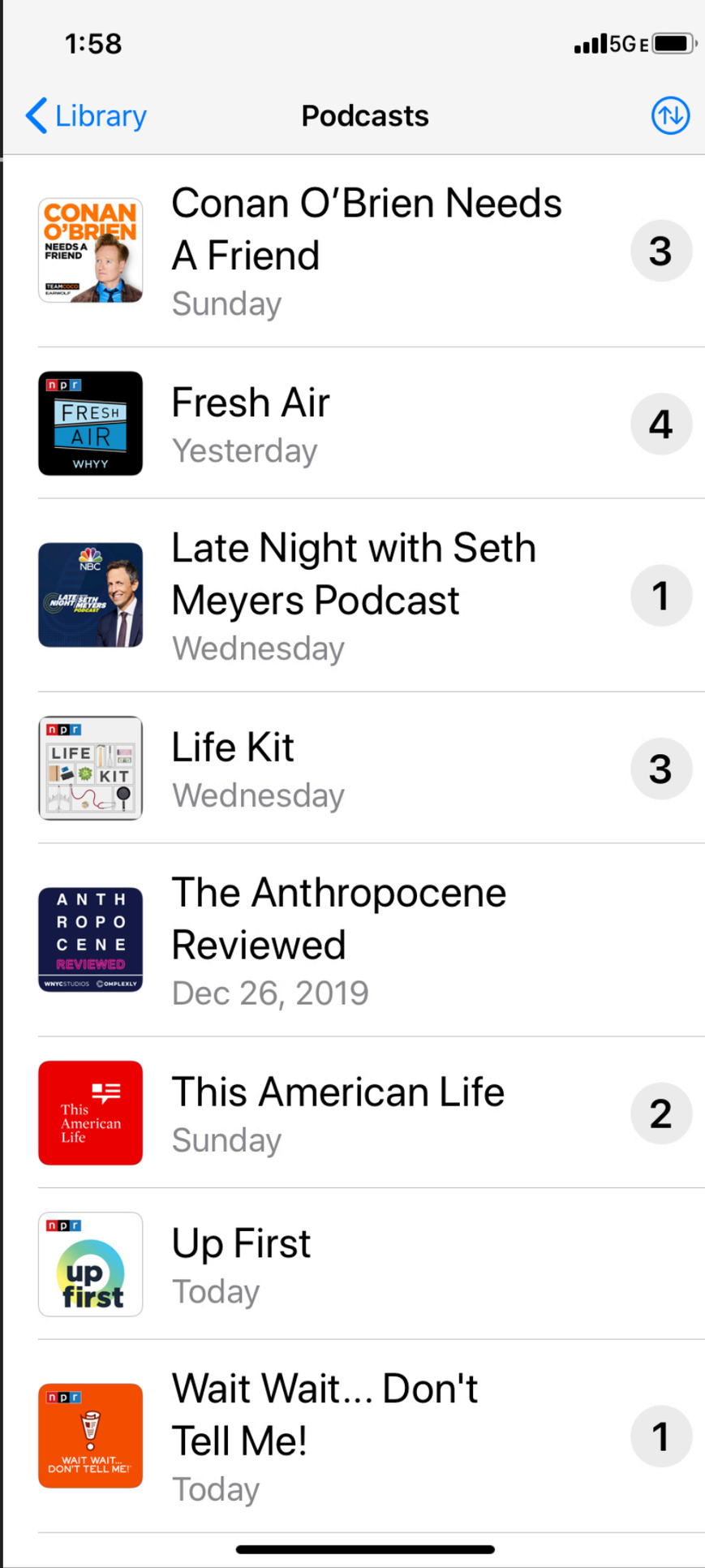


NAVIGATION CONTROLLER

- The navigation controller maintains an ordered array of child view controllers
  - This is called the navigation stack
  - Root view controller is at the bottom
  - Currently visible controller is at the top

NAVIGATION BAR

- Navigation controllers also manage a navigation bar, consisting of:
  - Left bar button item(s)
  - Title view
  - Right bar button item(s)



Castaway

NAVIGATION BAR

- Child view controllers tell the navigation controller what to display in the navigation bar
  - This is done through UIViewController's `navigationItem` and `title` properties

UIViewController

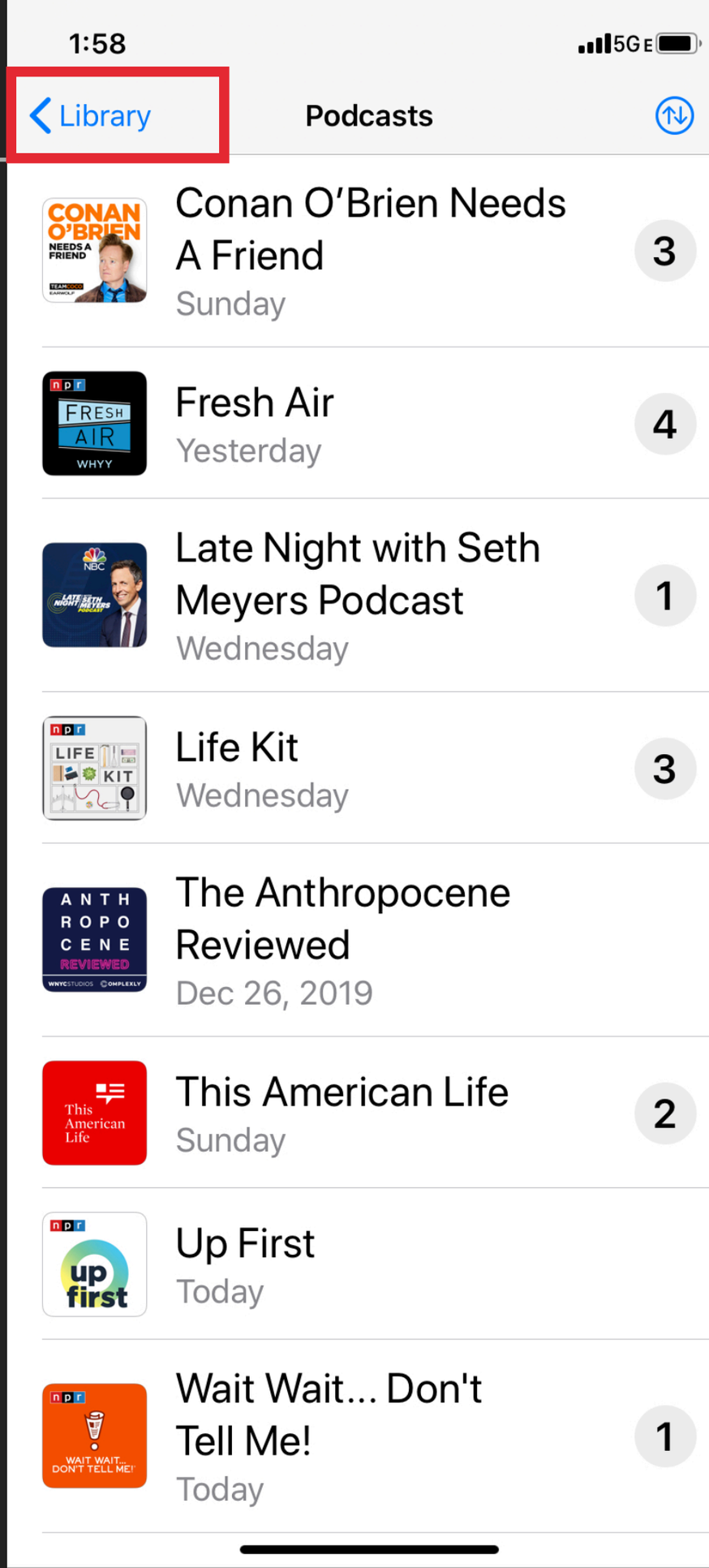
```
var navigationItem: UINavigationItem?

var title: String?

...
```

LEFT BAR BUTTON ITEM

- Typically provides navigation back to the previous view controller
  - Instance of `UIBarButtonItem`
- By default, displays the title of the previous view controller

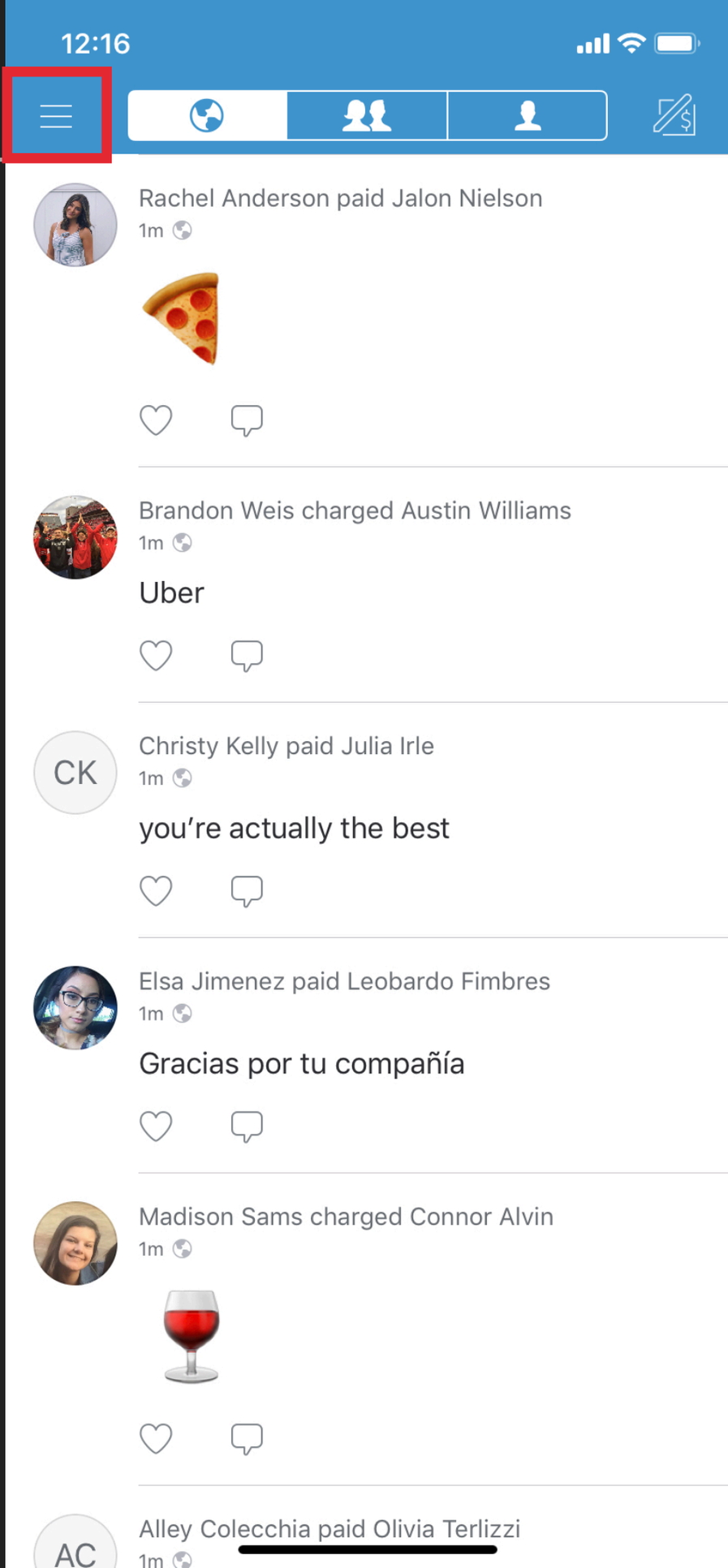


LEFT BAR BUTTON ITEM

- To customize the left bar button item…
  - On the current view controller, set:
    - `navigationItem.leftBarButtonItem` -OR-
    - `navigationItem.leftBarButtonItems`
  - On the previous view controller, set:
    - `navigationItem.backBarButtonItem`

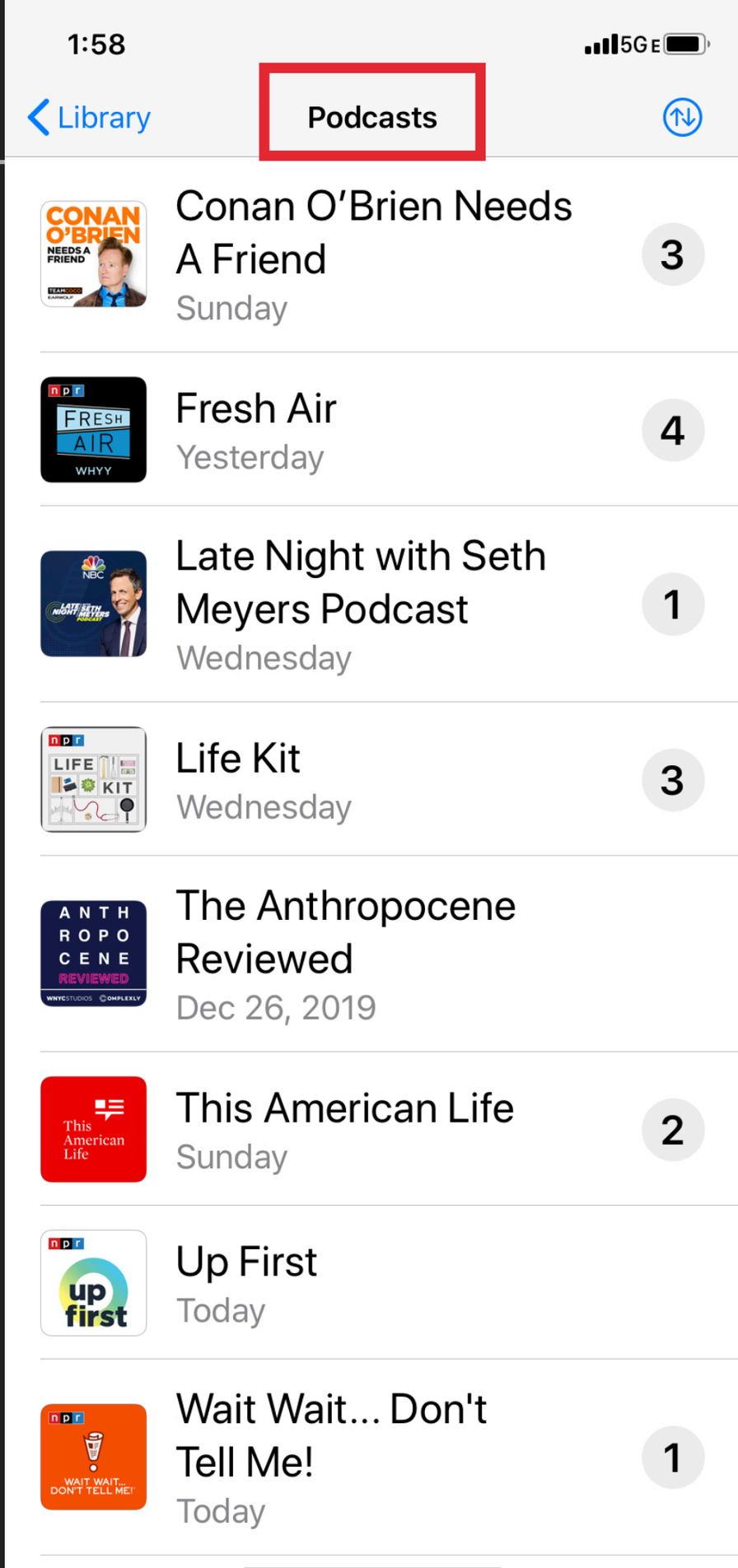
LEFT BAR BUTTON ITEM

- Example of a custom left bar button item



TITLE VIEW

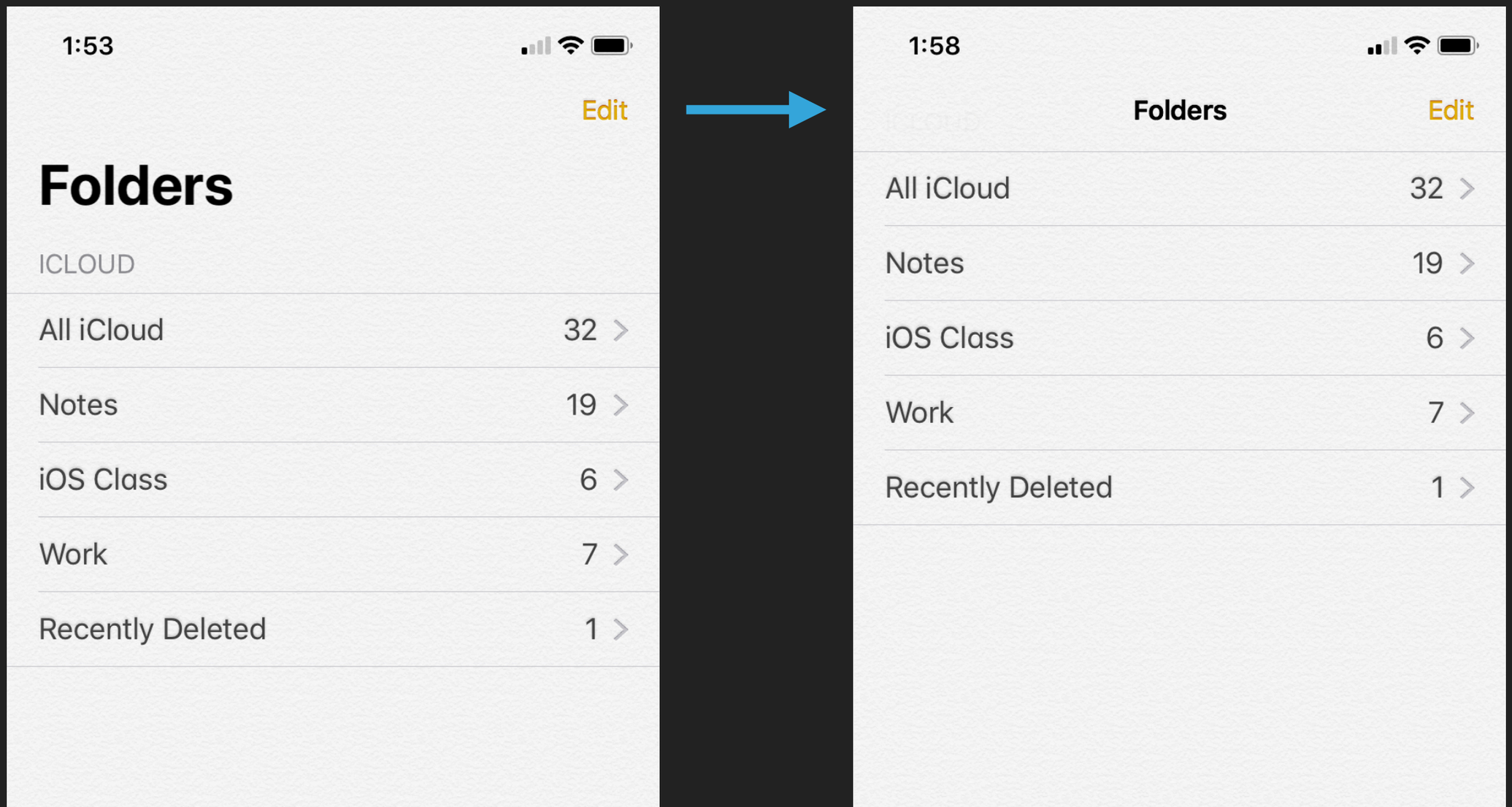
- By default, displays a label with the view controller's `title` property



Castaway

LARGE TITLES

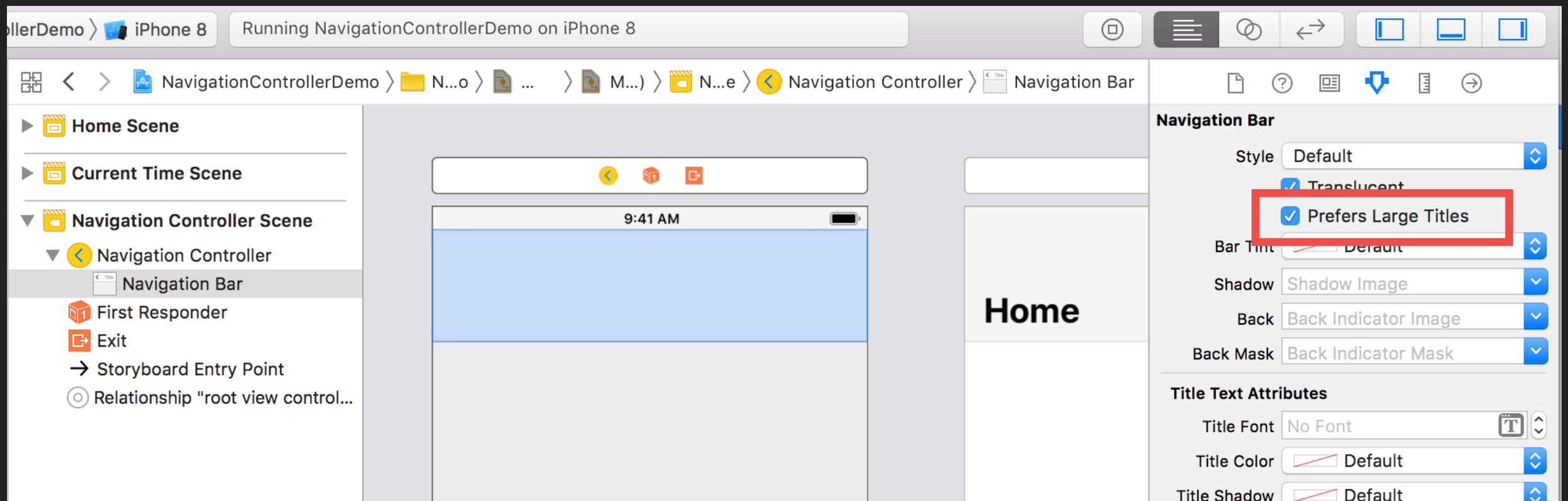
- Large titles were introduced in iOS 11



LARGE TITLES

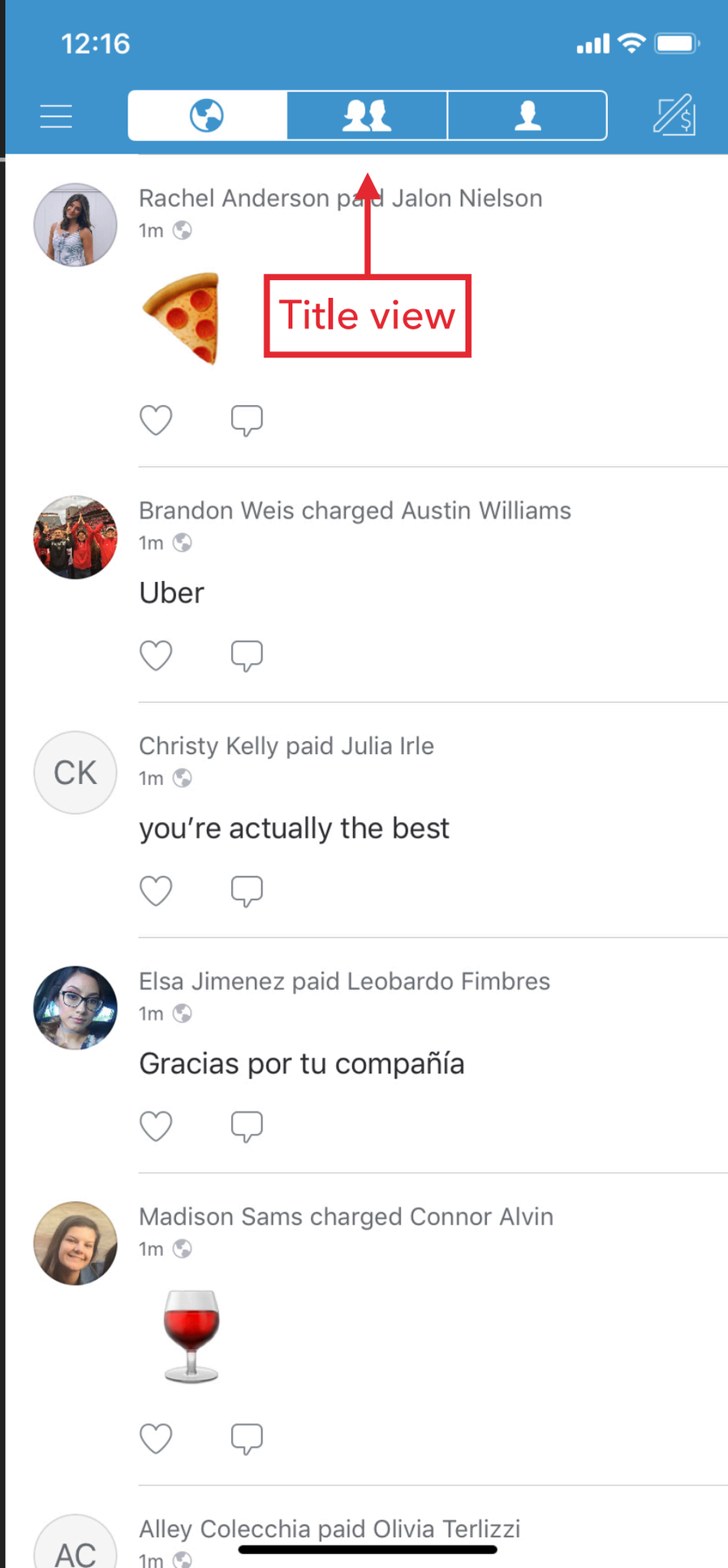
- Select navigation bar and check “Prefers Large Titles”
- Can also be done programmatically:

```
navigationController?.navigationBar.prefersLargeTitles = true
```



TITLE VIEW

- You can provide a custom title view by setting `navigationItem.titleView`



RIGHT BAR BUTTON ITEM

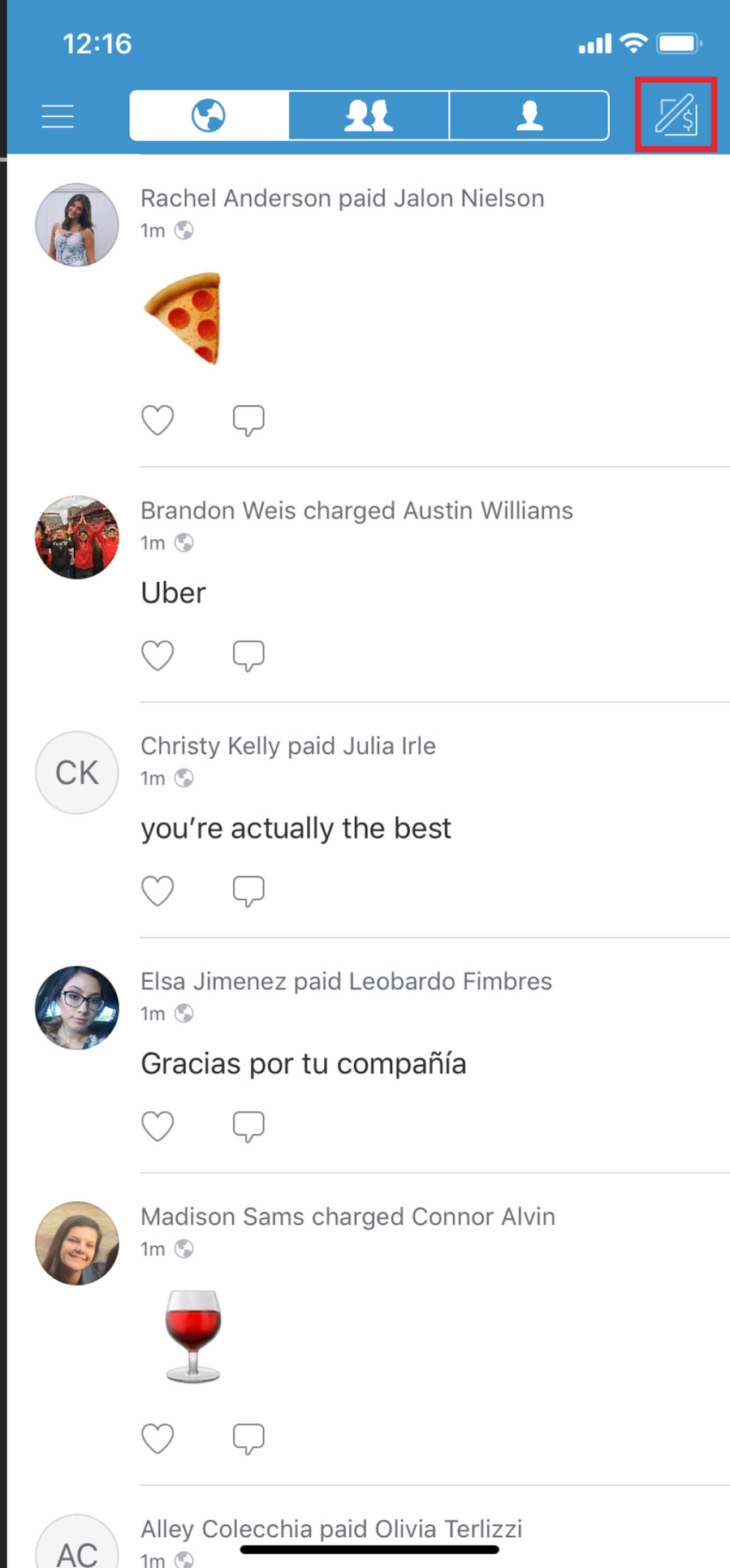
- Set right bar button item(s) with:

`navigationItem.rightBarButtonItem`

-OR-

`navigationItem.rightBarButtonItems`

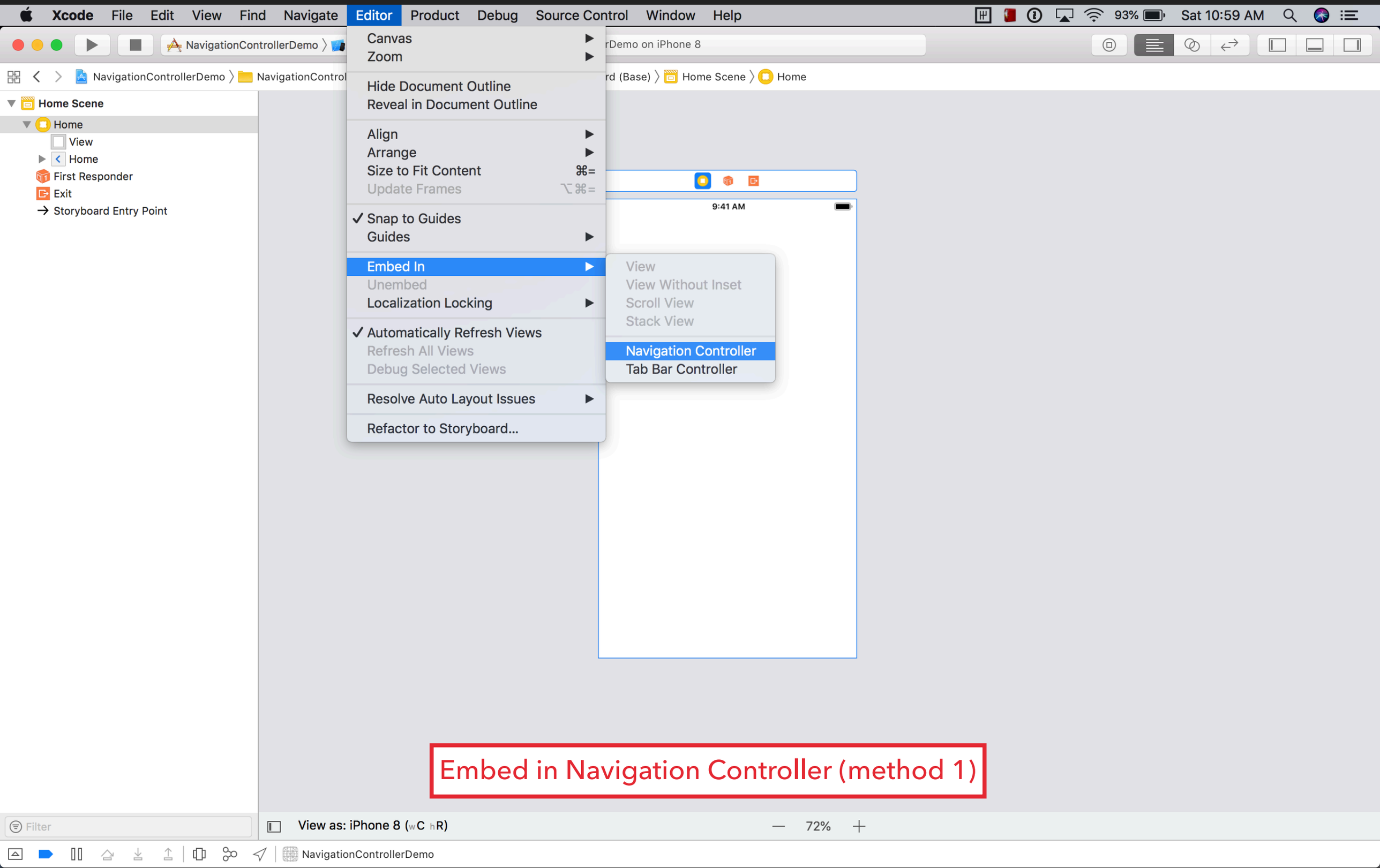
Venmo



CREATING A NAVIGATION STACK

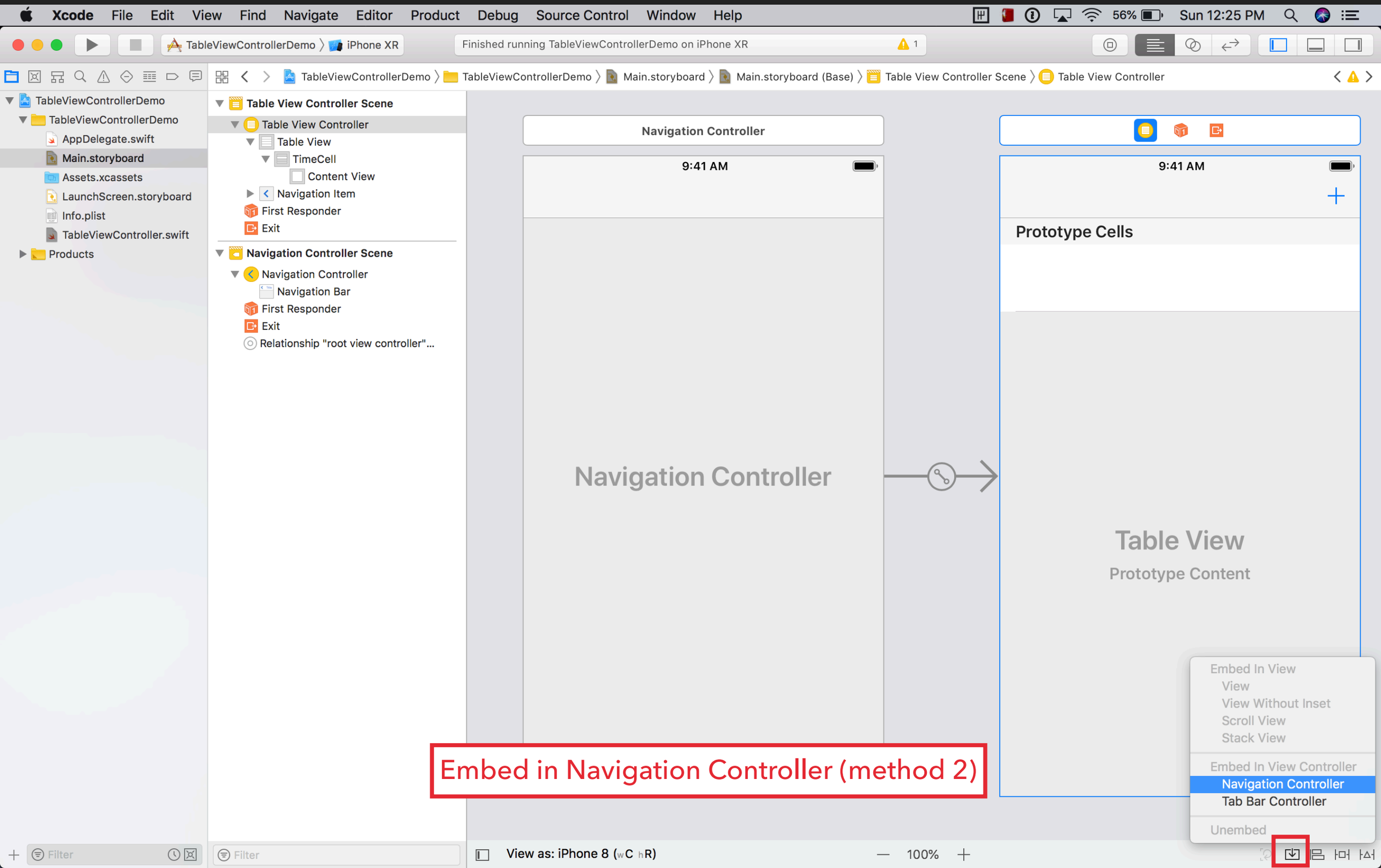
- The easiest way to create a navigation stack is using storyboard
- Segues* establish relationships and transitions between view controllers

*Pronounced “segway”.



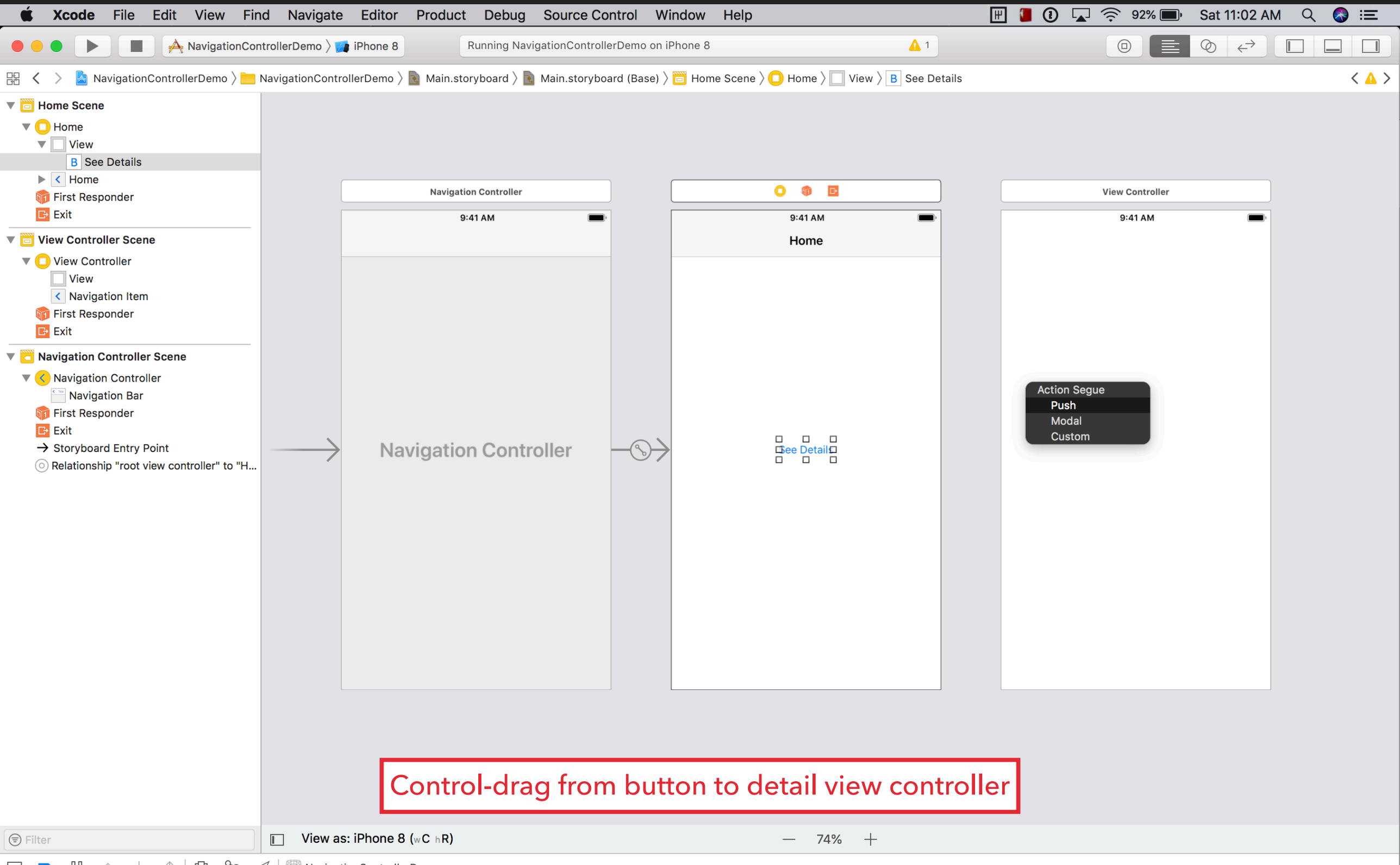
Embed in Navigation Controller (method 1)

NAVIGATION CONTROLLER

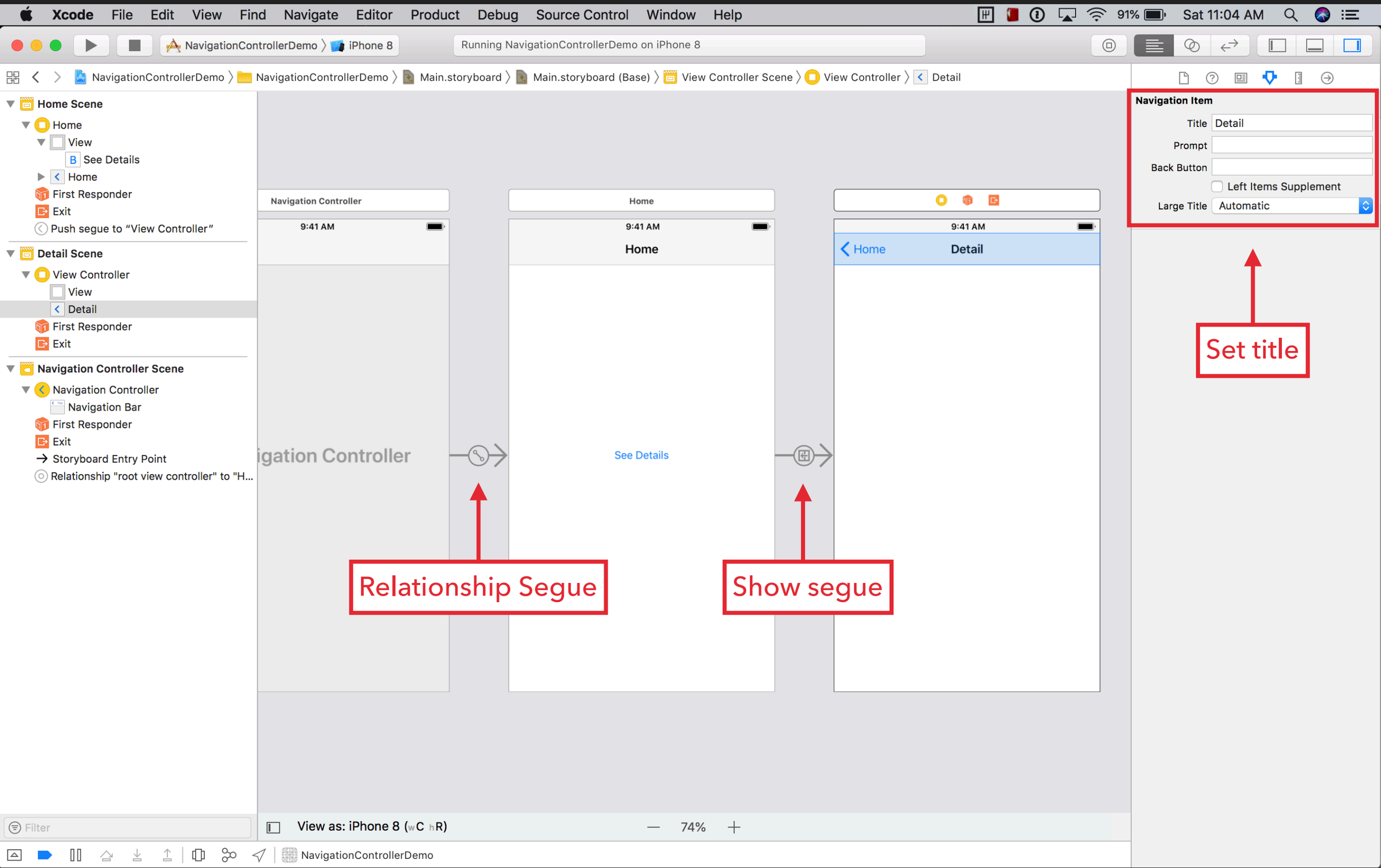


Embed in Navigation Controller (method 2)

NAVIGATION CONTROLLER



NAVIGATION CONTROLLER



PREPARE FOR SEGUE

- Override UIViewController's "prepare for segue" method to pass data between view controllers before the transition occurs

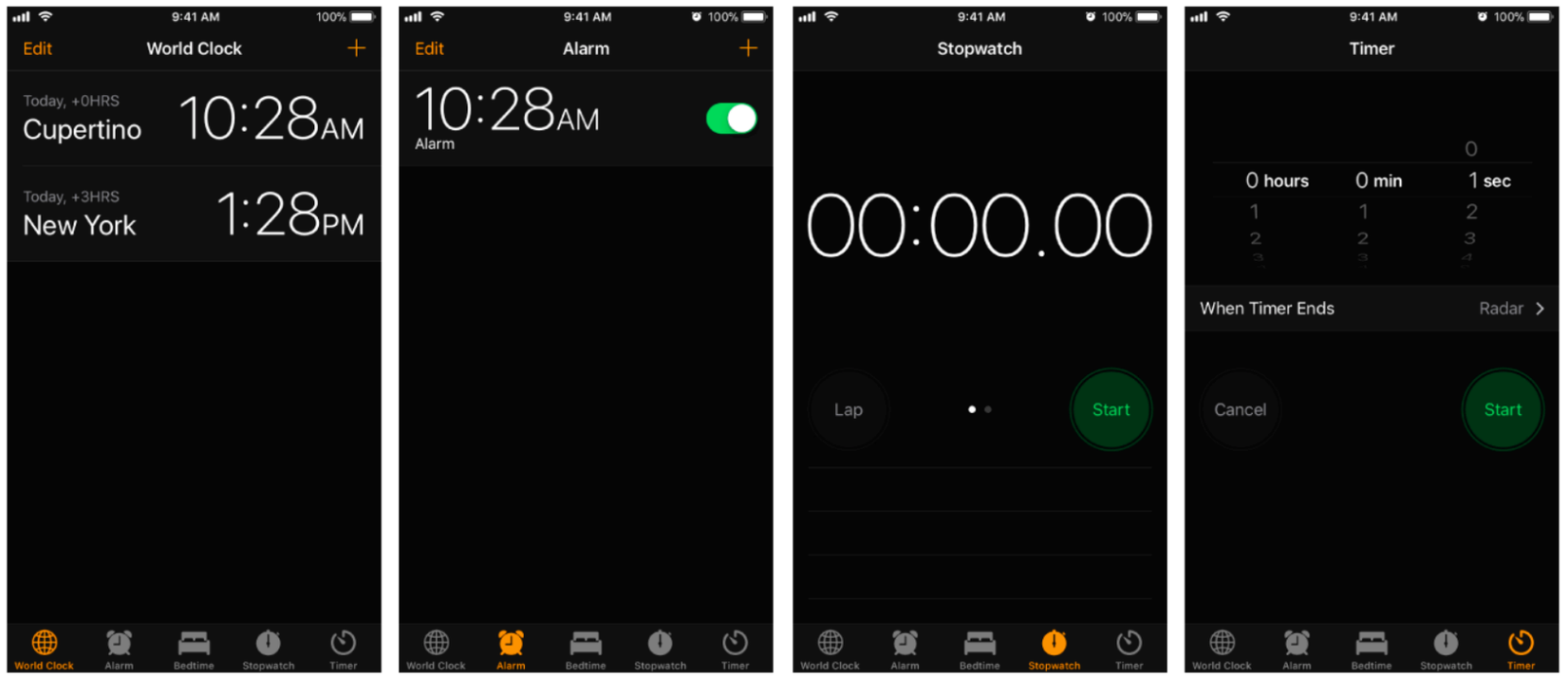
```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    let destination = segue.destination as? TimeViewController
    let formatter = DateFormatter()
    formatter.timeStyle = .long
    destination?.currentTime = formatter.string(from: Date())
}
```

TAB BAR CONTROLLER

TAB BAR CONTROLLER

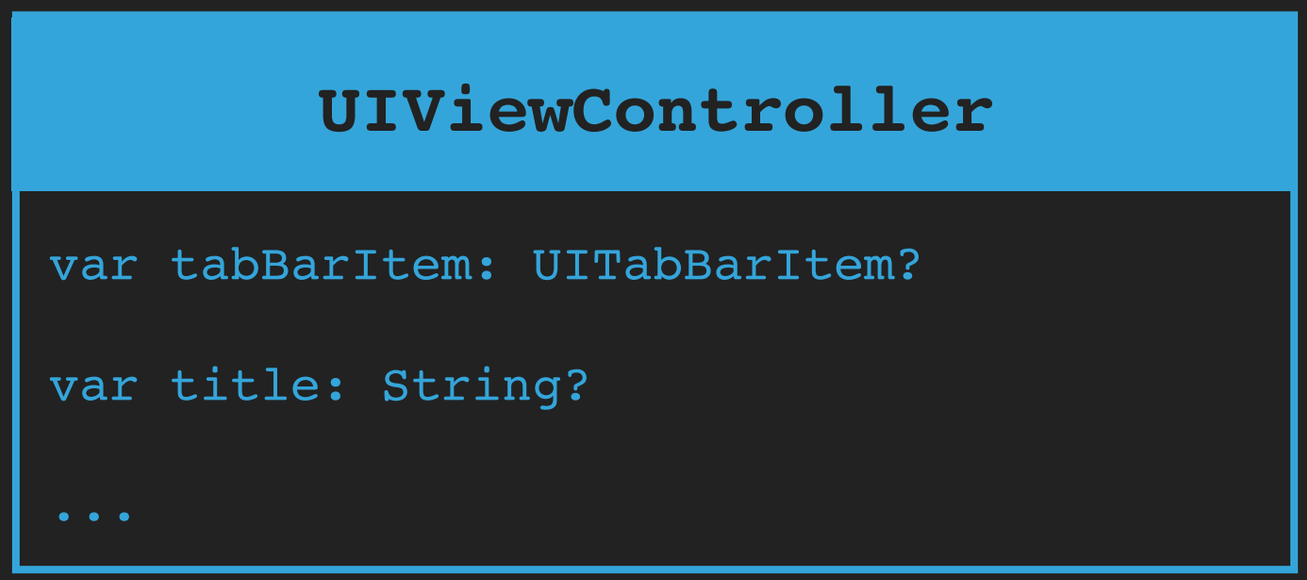
- A container view controller that manages a radio-style selection interface, where the selection determines which child view controller to display.

Figure 1 The tab bar interface in the Clock app



TAB BAR

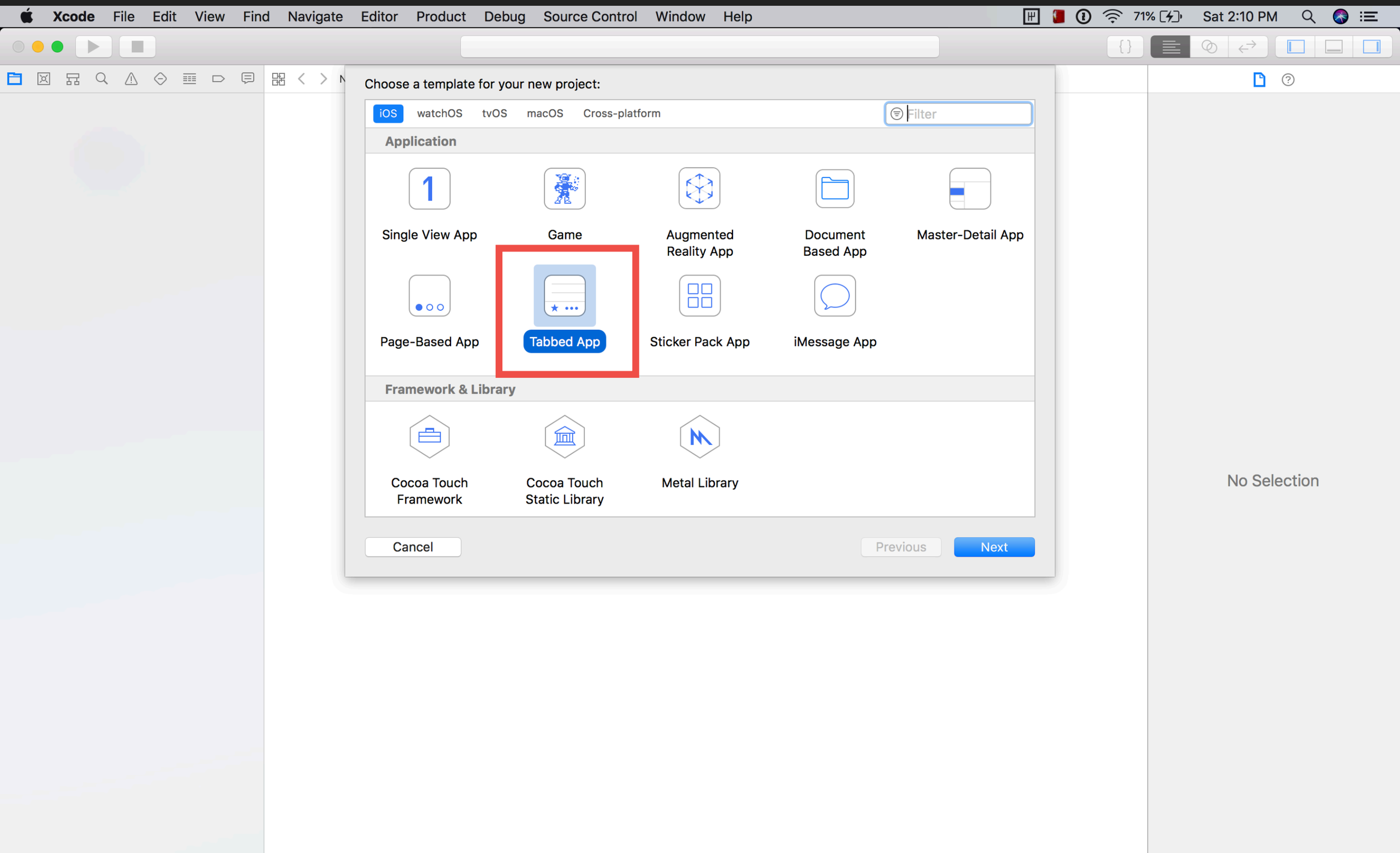
- Child view controllers tell the tab bar controller what to display in the tab bar
    - This is done through UIViewController's `tabBarItem` and `title` properties



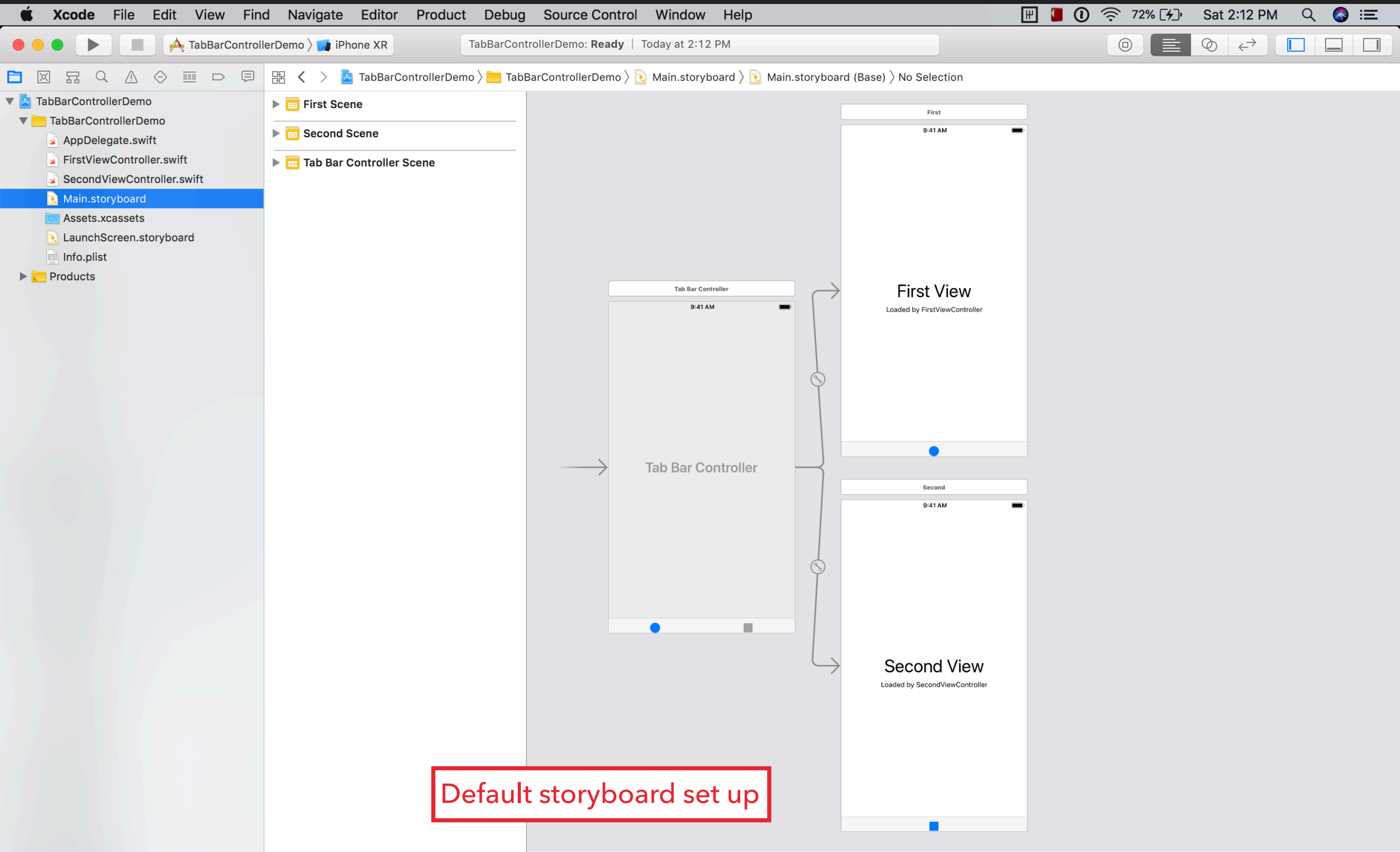
TAB BAR ITEM

- Each `UITabBarItem` can have a title and image
    - Use `UITabBarItemAppearance` to specify whether title and image should be stacked vertically or side-by-side

TAB BAR CONTROLLER



TAB BAR CONTROLLER



TAB BAR ITEMS

- “More” tab will automatically be displayed if you have more than 5 tabs

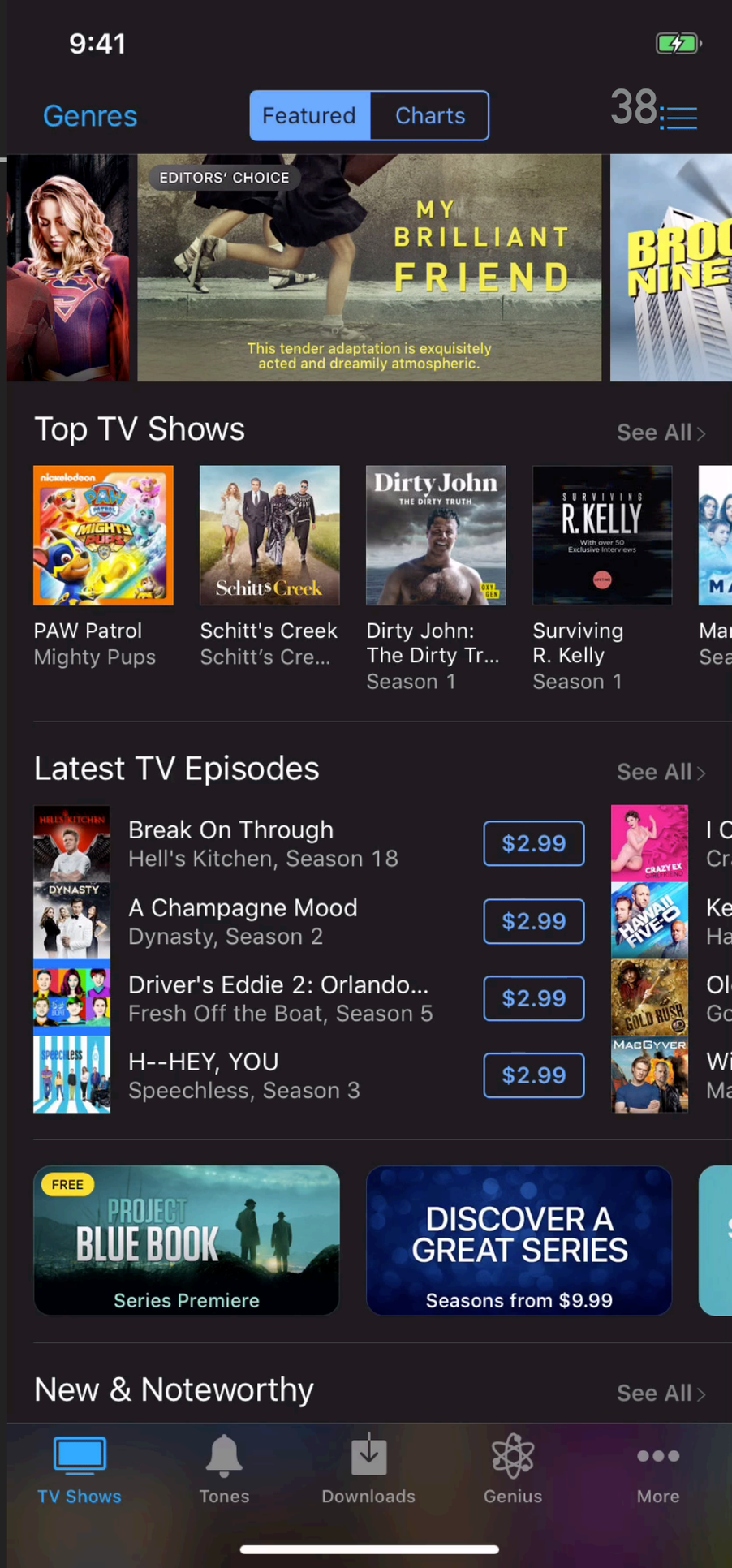
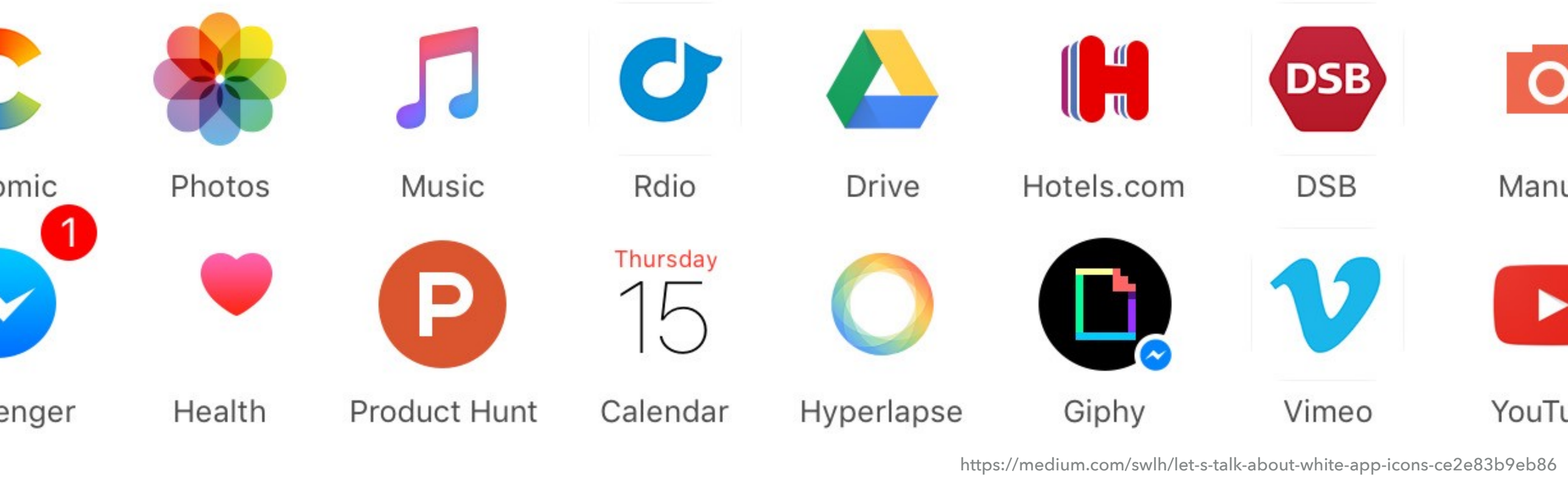


TABLE VIEW CONTROLLER



https://medium.com/swlh/let-s-talk-about-white-app-icons-ce2e83b9eb86

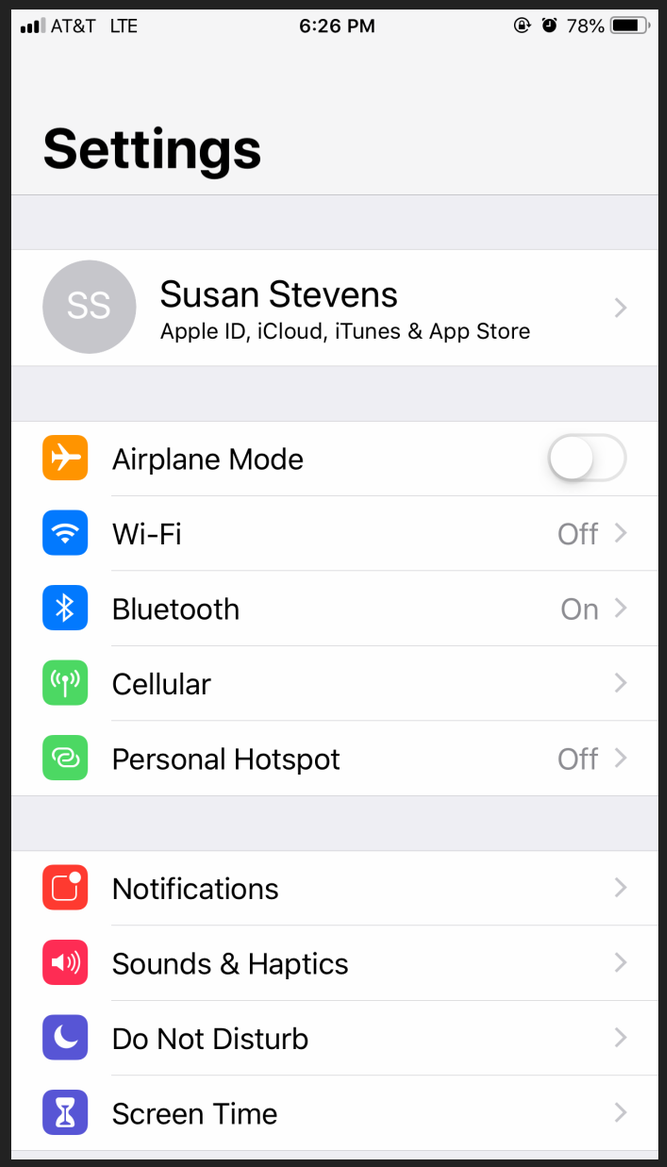
THERE ARE 2 MILLION+ APPS IN THE APP STORE

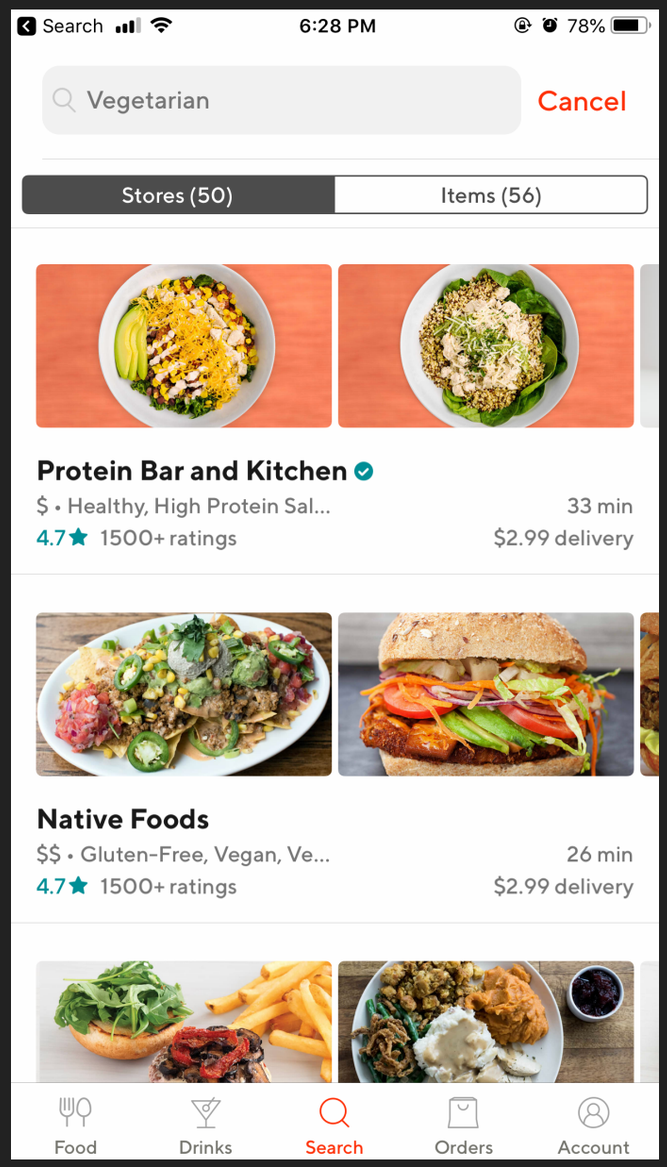
WHAT DO THEY HAVE IN COMMON?

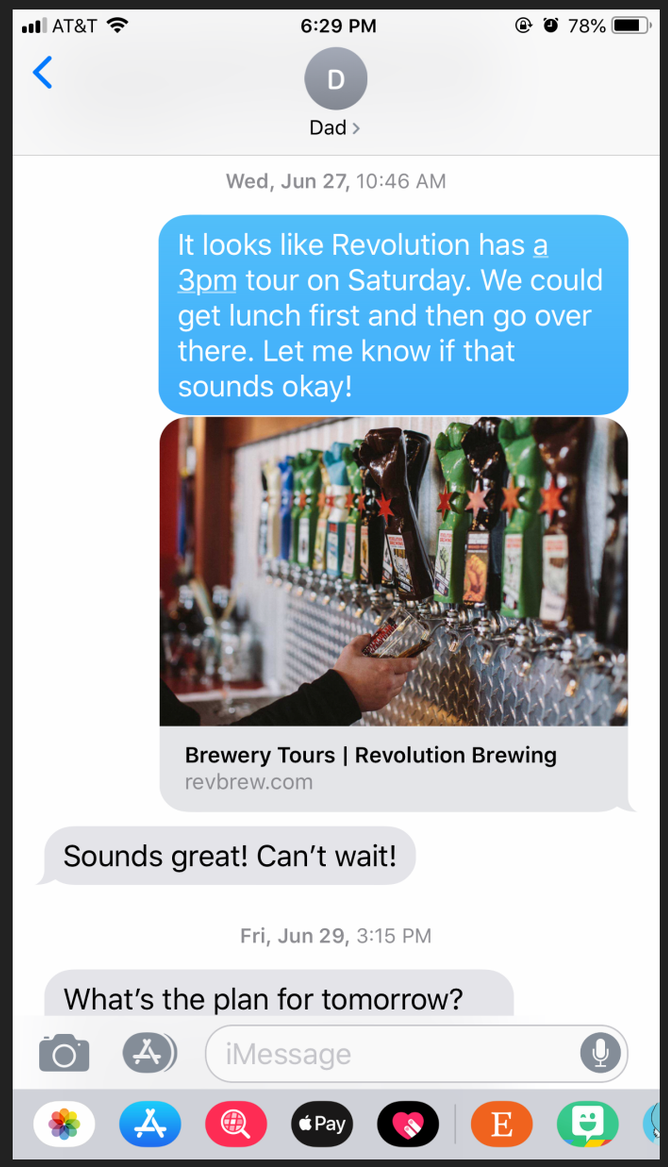
THEY ALL HAVE TABLE VIEWS

(Okay, almost all.)

WHAT IS A TABLE VIEW?

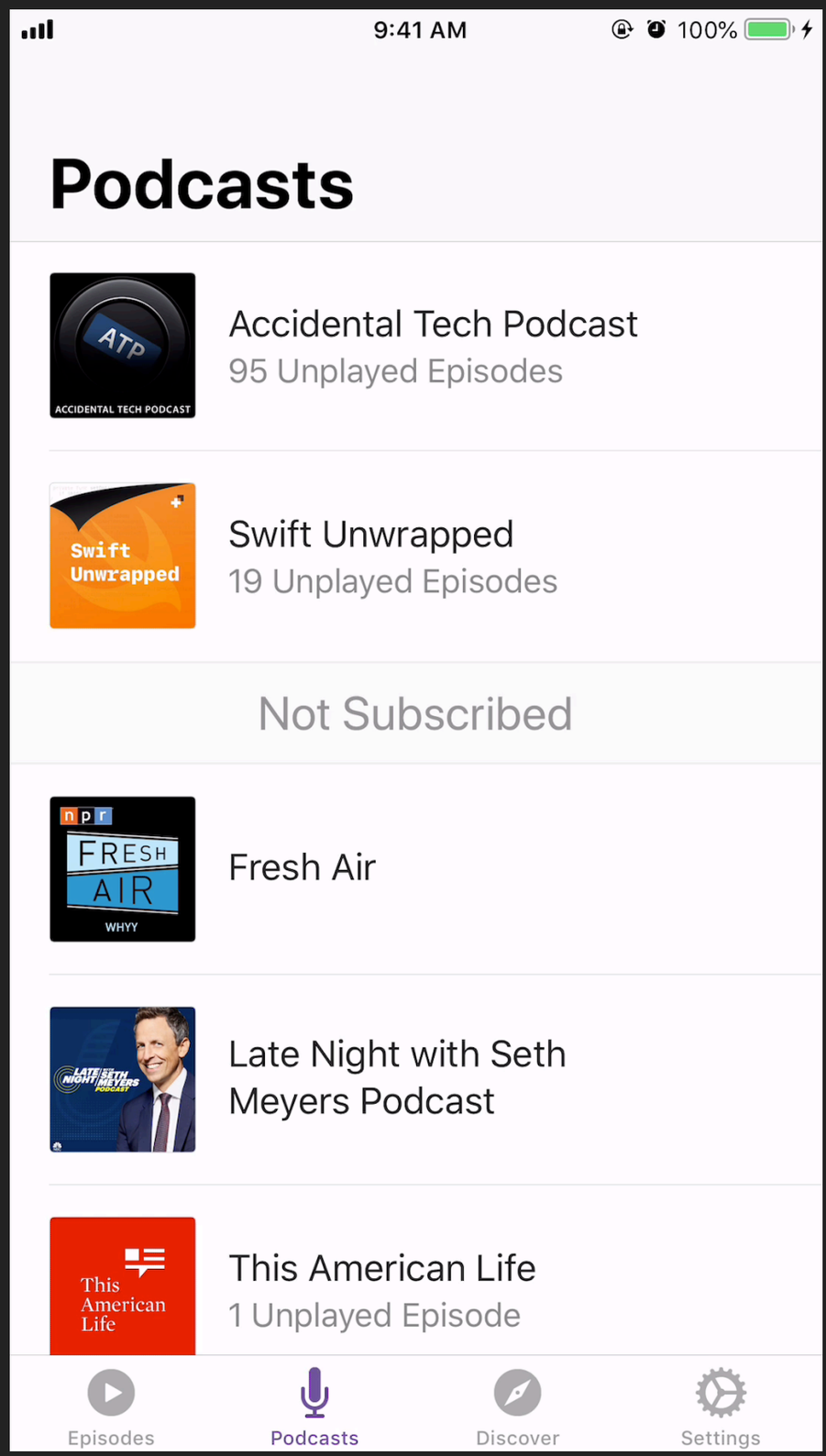






WHAT IS A TABLE VIEW?

- If it has rows that scroll vertically, it's probably a `UITableView`
- Beyond that, UITableViews are infinitely customizable



HOW DOES IT WORK?

- Apple does all of the hard work (scrolling, rendering) so that you can do the fun stuff. 😀
- You conform to two protocols:
  - `UITableViewDataSource`
  - `UITableViewDelegate`

TABLE VIEW DATA SOURCE

- At minimum, `UITableView` needs to know three things:
  - How many sections should it have?
  - How many rows in each section?
  - Which cell should be displayed in each row?

TABLE VIEW DATA SOURCE

- You answer these questions with three `UITableViewDataSource` methods

```swift
24
25      override func numberOfSections(in tableView: UITableView) -> Int {
26          return 1
27      }
28
29      override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
30          return 20
31      }
32
33      override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
34          let cell = tableView.dequeueReusableCell(withIdentifier: "reuseIdentifier", for: indexPath)
35
36          // Configure the cell...
37
38          return cell
39      }
```

TABLE VIEW DATA SOURCE

- Imagine a table view with thousands of rows.
- Would you want to load every row into memory as soon as the table view appears on screen?

TABLE VIEW DATA SOURCE

- Instead of creating every cell, table views maintain only the cells they need at a given time
- As the user scrolls, some cells go off screen and others come on screen
- Rather than creating entirely new cells, table views reuse cells as the user scrolls

TABLE VIEW DATA SOURCE

- The table view will ask the data source for a cell to display at a particular index path (row and section)

```
33      override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
34          let cell = tableView.dequeueReusableCell(withIdentifier: "reuseIdentifier", for: indexPath)
35
36          // Configure the cell...
37
38          return cell
39      }
```

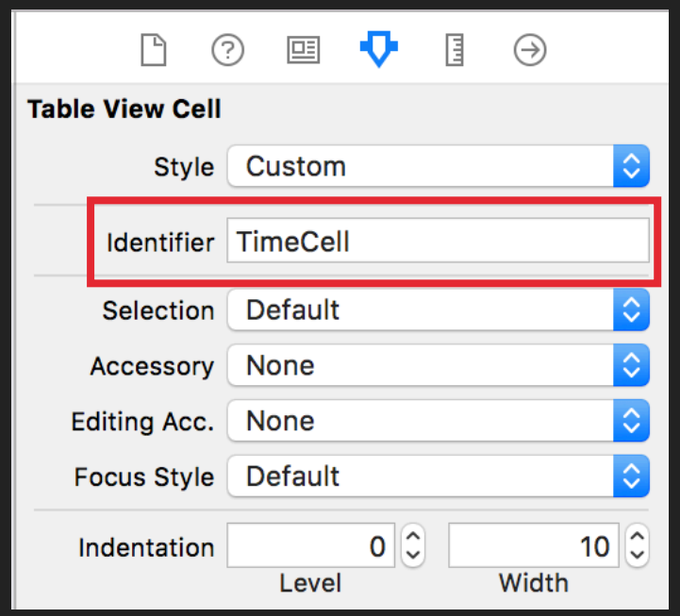
TABLE VIEW DATA SOURCE

- The data source gets a reusable cell from the table view using the reuse identifier

```
33      override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
34          let cell = tableView.dequeueReusableCell(withIdentifier: "reuseIdentifier", for: indexPath)
35
36          // Configure the cell...
37
38          return cell
39      }
```

TABLE VIEW DATA SOURCE

- Remember to set the reuseIdentifier in storyboard




```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "TimeCell", for: indexPath)
    cell.textLabel?.text = timeStrings[indexPath.row]
    return cell
}
```

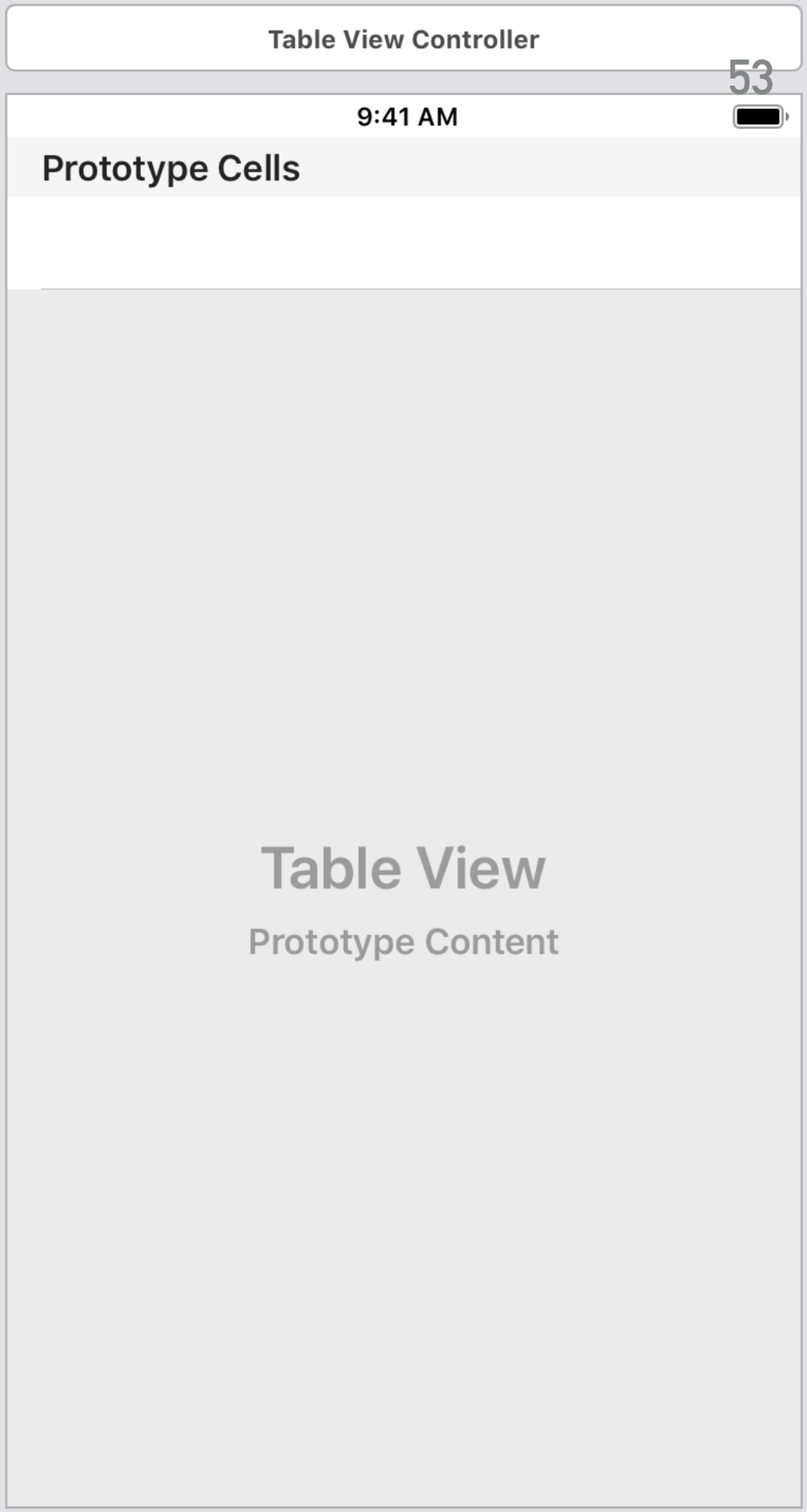
TABLE VIEW DATA SOURCE

- The data source configures the cell for the given index path and then returns it

```
33      override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
34          let cell = tableView.dequeueReusableCell(withIdentifier: "reuseIdentifier", for: indexPath)
35
36          // Configure the cell...
37
38          return cell
39      }
```

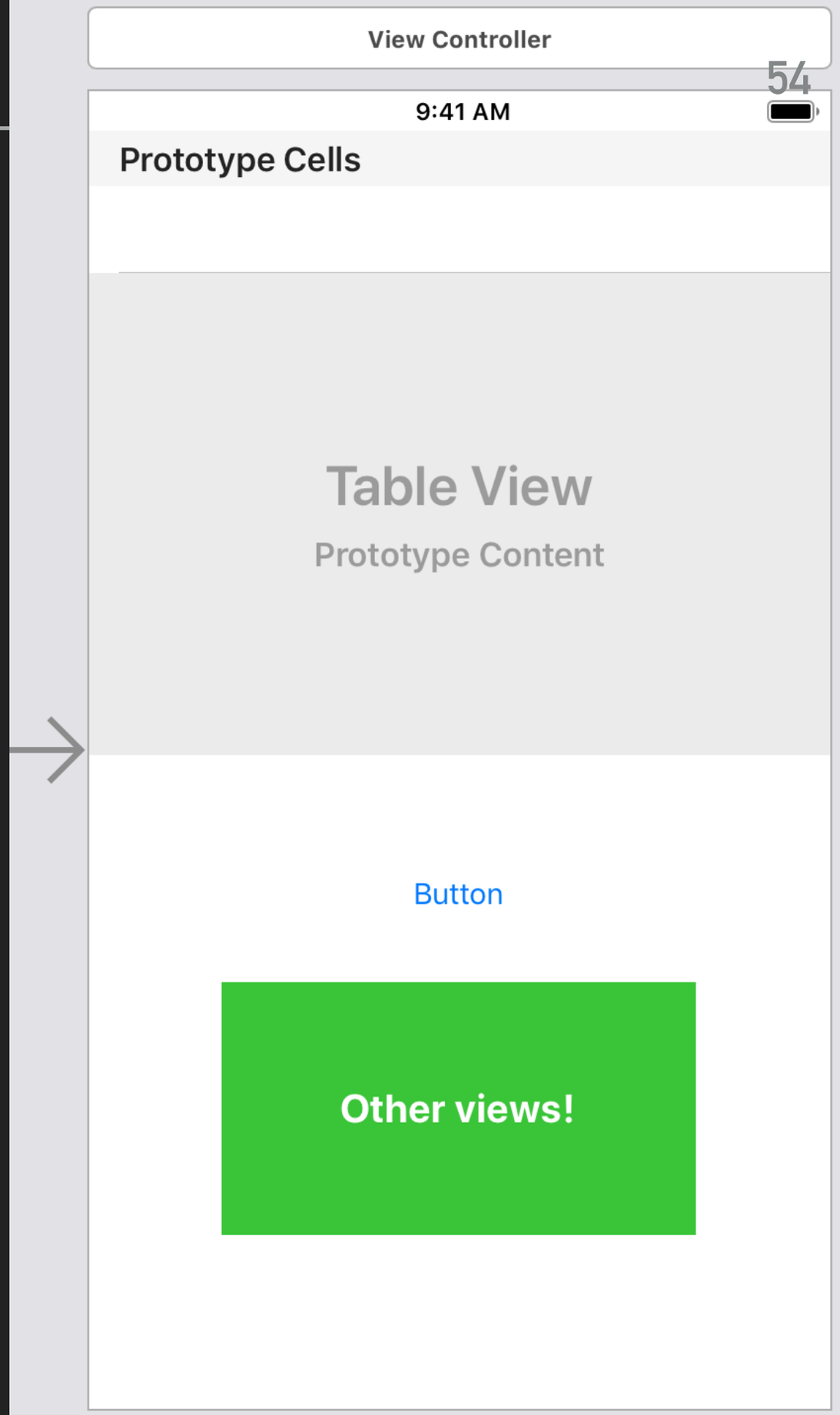
TABLE VIEW CONTROLLER

- Requires less set-up
- Built-in support for editing and refreshing
- Table view fills the entire view controller



VIEW CONTROLLER + TABLE VIEW

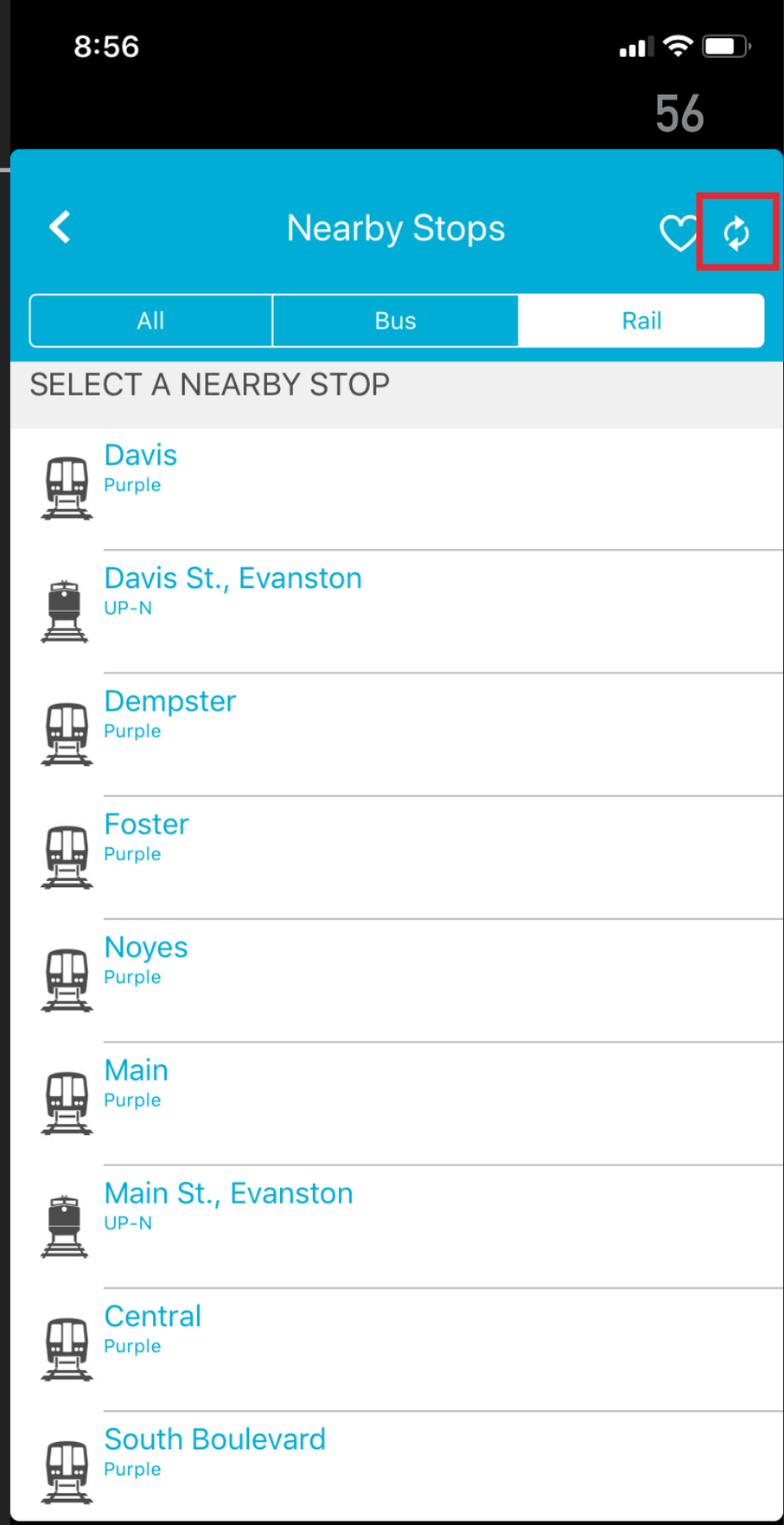
- Requires more set-up
- Table view does not necessarily fill the entire view controller



PULL-TO-REFRESH

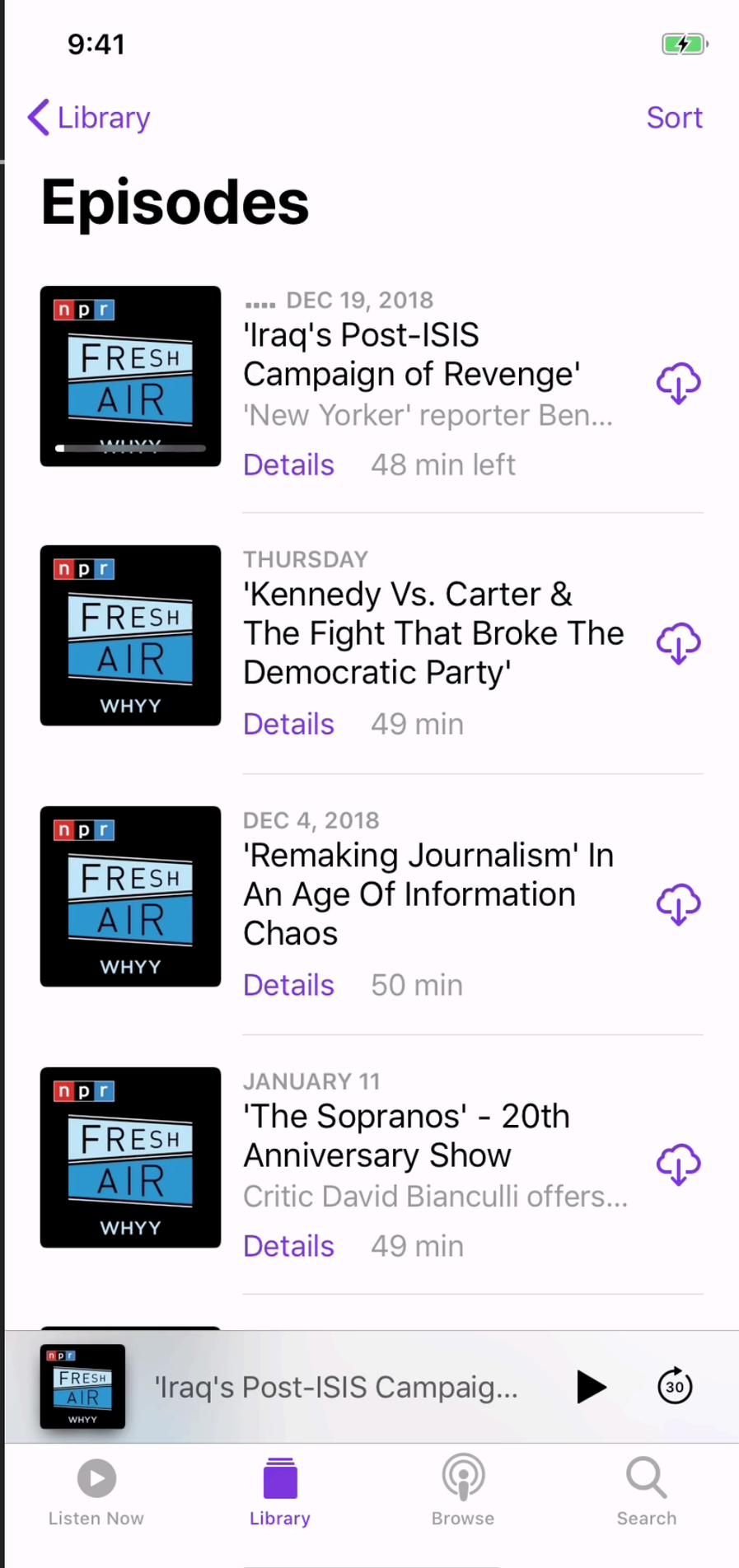
REFRESH BUTTON

- This is the old method of allowing users to refresh a page
- Refresh button takes up valuable real estate in the navigation bar



PULL-TO-REFRESH

- Invented by Loren Brichter during the development of Tweetie
- Now a built-in component



A BRIEF INTRODUCTION TO

NETWORKING

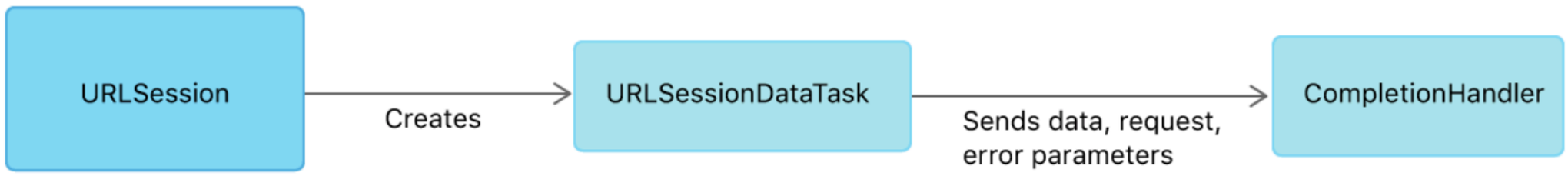
URLSESSION

- Almost all apps need to fetch data from the network
- This can be achieved using `URLSession` and related classes

URLSESSION

- Create a `URLSessionDataTask` by providing:
  - The URL for the resource you want to fetch
  - A completion handler (closure) that will be called when the data is available

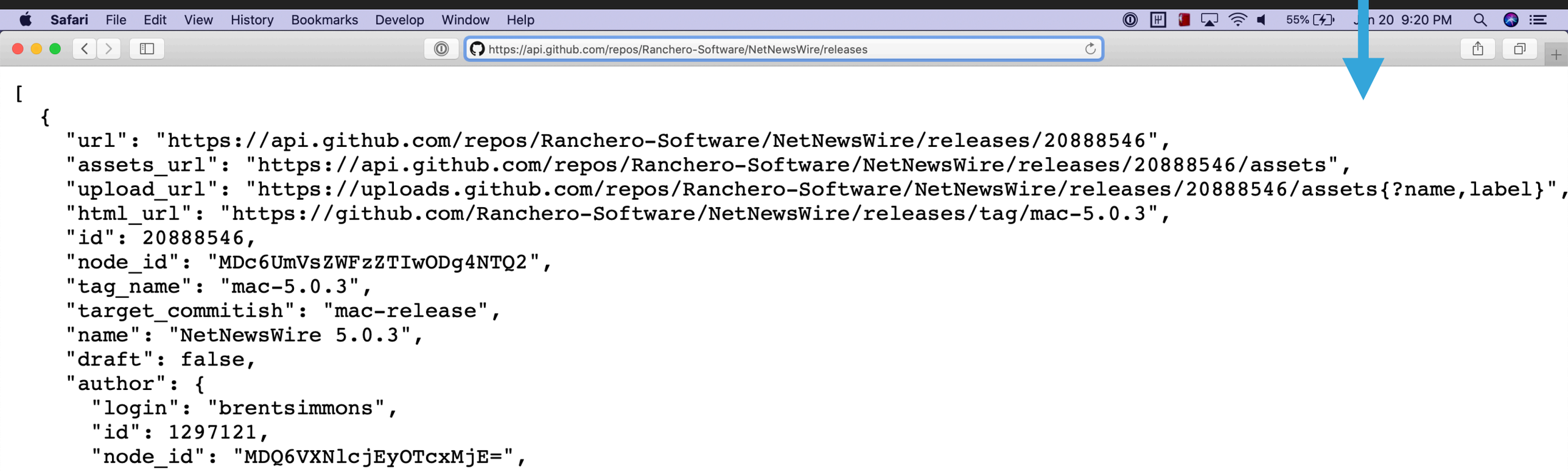
Figure 1 Creating a completion handler to receive results from a task



URLSESSION

- It's common for URLs to return data as JSON
- The app must parse the JSON into Swift model objects

GitHub API



JSON TO SWIFT

- `Decodable` is a protocol that indicates that a new instance of the type can be created by decoding JSON
    - Properties that are present in the JSON, but not the model are ignored
    - Properties that are present in the model, but potentially absent from the JSON must be optional

GITHUB API MODELS

```
2
3   struct Release: Decodable {
4       let name: String
5       let createdAt: String
6       let author: Author
7   }
8
9   struct Author: Decodable {
10      let login: String
11  }
12
```

GITHUB CLIENT

- `GitHubClient` is responsible for:
  - Fetching JSON from the GitHub API's /releases endpoint
  - Parsing the JSON into Swift model objects

```
14
15  class GitHubClient {
16
17      static func fetchReleases(completion: @escaping ([Release]?, Error?) -> Void) {
18          // TODO: - fetch a list of releases
19      }
20  }
21
```

GITHUB CLIENT

- The `static` keyword indicates that this method lives on the type itself (not an instance of the type)

```
14
15  class GitHubClient {
16
17      static func fetchReleases(completion: @escaping ([Release]?, Error?) -> Void) {
18          // TODO: - fetch a list of releases
19      }
20  }
21
```

GITHUB CLIENT

- `fetchReleases` takes a closure called `completion` as a parameter

```
14
15  class GitHubClient {
16
17      static func fetchReleases(completion: @escaping ([Release]?, Error?) -> Void) {
18          // TODO: - fetch a list of releases
19      }
20  }
21
```

GITHUB CLIENT

- `@escaping` means this closure may be execute after the `fetchReleases` method returns

```
14
15  class GitHubClient {
16
17      static func fetchReleases(completion: @escaping ([Release]?, Error?) -> Void) {
18          // TODO: - fetch a list of releases
19      }
20  }
21
```

GITHUB CLIENT

- First, we create a variable for the URL

```
14
15  class GitHubClient {
16
17      static func fetchReleases(completion: @escaping ([Release]?, Error?) -> Void) {
18          let url = URL(string: "https://api.github.com/repos/Ranchero-Software/NetNewsWire/releases")!
19
20          // ...
21
22
```

GITHUB CLIENT

- Next, we create a `URLSessionDataTask` to fetch data from the URL

```
14
15  class GitHubClient {
16
17      static func fetchReleases(completion: @escaping ([Release]?, Error?) -> Void) {
18          let url = URL(string: "https://api.github.com/repos/Ranchero-Software/NetNewsWire/releases")!
19
20          let task = URLSession.shared.dataTask(with: url) { data, _, error in
21
```

GITHUB CLIENT

- Inside the data task's closure, `guard` that data has a value and error is nil
- If one of these is not the case, call the `fetchReleases` closure with nil and the error

```
14
15  class GitHubClient {
16
17      static func fetchReleases(completion: @escaping ([Release]?, Error?) -> Void) {
18          let url = URL(string: "https://api.github.com/repos/Ranchero-Software/NetNewsWire/releases")!
19
20          let task = URLSession.shared.dataTask(with: url) { data, _, error in
21
22              guard let data = data, error == nil else {
23                  DispatchQueue.main.async { completion(nil, error) }
24                  return
25              }
26
27              do {
```

GITHUB CLIENT

- Use `JSONDecoder` to decode the JSON into an array of `Release` objects
- This can throw an error, so we use a do-try-catch block

```
22          guard let data = data, error == nil else {
23              DispatchQueue.main.async { completion(nil, error) }
24              return
25          }
26
27          do {
28              let decoder = JSONDecoder()
29              decoder.keyDecodingStrategy = .convertFromSnakeCase
30              let releases = try decoder.decode([Release].self, from: data)
31              DispatchQueue.main.async { completion(releases, nil) }
32          } catch (let parsingError) {
33              DispatchQueue.main.async { completion(nil, parsingError) }
34          }
35      }
```

GITHUB CLIENT

- Call `resume()` to start the task
  - It's created in the "paused" state

```
22              guard let data = data, error == nil else {
23                  DispatchQueue.main.async { completion(nil, error) }
24                  return
25              }
26
27              do {
28                  let decoder = JSONDecoder()
29                  decoder.keyDecodingStrategy = .convertFromSnakeCase
30                  let releases = try decoder.decode([Release].self, from: data)
31                  DispatchQueue.main.async { completion(releases, nil) }
32              } catch (let parsingError) {
33                  DispatchQueue.main.async { completion(nil, parsingError) }
34              }
35          }
36
37          task.resume()
38      }
39  }
40
```

GITHUB CLIENT

```
14
15  class GitHubClient {
16
17      static func fetchReleases(completion: @escaping ([Release]?, Error?) -> Void) {
18          let url = URL(string: "https://api.github.com/repos/Ranchero-Software/NetNewsWire/releases")!
19
20          let task = URLSession.shared.dataTask(with: url) { data, _, error in
21
22              guard let data = data, error == nil else {
23                  DispatchQueue.main.async { completion(nil, error) }
24                  return
25              }
26
27              do {
28                  let decoder = JSONDecoder()
29                  decoder.keyDecodingStrategy = .convertFromSnakeCase
30                  let releases = try decoder.decode([Release].self, from: data)
31                  DispatchQueue.main.async { completion(releases, nil) }
32              } catch (let parsingError) {
33                  DispatchQueue.main.async { completion(nil, parsingError) }
34              }
35          }
36
37          task.resume()
38      }
39  }
40
```

USING THE GITHUB CLIENT

```
40
41  GitHubClient.fetchReleases { (releases, error) in
42      guard let releases = releases, error == nil else {
43          print(error!)
44          return
45      }
46
47      for release in releases {
48          print("""
49                  ✔ \(release.name)
50                  Author: \(release.author.login)
51                  Date: \(release.createdAt)
52
53                  """)
54      }
    }
56
```

UPDATES TO THE UI SHOULD ALWAYS HAPPEN ON THE MAIN QUEUE

Remember this for interviews.

ASSIGNMENT #3

GITHUB ISSUES

GITHUB ISSUES

- Build an app that lists open and closed GitHub issues from an open-source repo
- We will use:
  - `UITabBarController`
  - `UINavigationController`
  - `UITableViewController`
  - `URLSession`

GITHUB ISSUES

- Due Wednesday, January 29 at 5:29pm
- Post any questions to the class forum
- We're getting into more complex material, but please refrain from using any 3rd-party frameworks

GITHUB ISSUES

- Remember to commit to the development branch and then open a pull request into master